T U M

I N S T I T U T F Ü R I N F O R M A T I K

Ausführungssemantik von AutoFocus-Modellen: Isabelle/HOL-Formalisierung und Äquivalenzbeweis

David Trachtenherz

TUM-I0903
Januar 09

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-01-I0903-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Ausführungssemantik von AUTOFOCUS-Modellen: Isabelle/HOL-Formalisierung und Äquivalenzbeweis

David Trachtenherz

22. Januar 2009

Zusammenfassung

Wir formalisieren den stark kausalen Ausschnitt der Ausführungssemantik des CASE-Werkzeugs AUTOFOCUS in dem Beweisassistenten Isabelle/HOL und zeigen die Äquivalenz dieser Darstellung und der formalen Definition in [12] sowie im Abschnitt 4.2.1.

Inhaltsverzeichnis

1 Einführung

Wir formalisieren den stark kausalen Ausschnitt der Ausführungssemantik des CASE-Werkzeugs AUTOFOCUS in dem Beweisassistenten Isabelle/HOL und zeigen die Äquivalenz dieser Isabelle/HOL-Darstellung und der Semantikdefinition in [12, Abschnitt 4.1.2] sowie im Abschnitt 4.2.1 dieses Berichts.

Das CASE-Werkzeug AUTOFOCUS [11, 6] ermöglicht die Modellierung hierarchisch aufgebauter Systeme. Ein AUTOFOCUS-Modell besteht aus Komponenten, die miteinander über gerichtete Kanäle kommunizieren. Jede Komponente besitzt entweder eine *Strukturverfeinerung* durch Teilkomponenten oder eine *Verhaltensverfeinerung* durch einen Zustandsautomaten.

Die Struktur eines Systems wird durch einen Baum hierarchisch aufgebauter Komponenten beschrieben. Die Knoten des Strukturbaums sind *SSDs (System Structure Diagrams)*. Ein SSD strukturiert eine Komponente in mehrere Teilkomponenten (Strukturverfeinerung) – in der Abb. 1 sind es beispielsweise `MainComponent` und `Component1`. Die Blätter des Strukturbaums sind Komponenten, deren Verhalten operational durch *STDs (State Transition Diagrams)* spezifiziert wird (Verhaltensverfeinerung) – wie z.B. `Automaton1_1` in der Abb. 1. Jedem Zustandsübergang in einem STD können Aktionen zugewiesen werden, die beim Schalten des Übergangs ausgelöst werden. Eine Aktion kann Ausgabe von Ergebnissen an Ausgabeports der Komponente sowie Code in der Sprache QuestF [2] enthalten – einer einfachen funktionalen Sprache, mit deren Hilfe zusätzliche Datentypen und Funktionen auf Datentypen in *DTDs (Data Type Definitions)* definiert werden können.

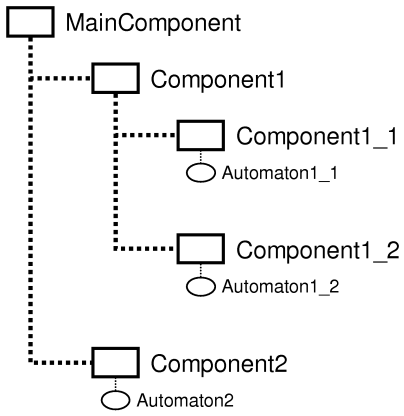

Abbildung 1: Baumstruktur eines AUTOFOCUS-Modells

Isabelle/HOL [9] ist ein generischer Beweisassistent für verschiedene Logiken, insbesondere die Prädikatenlogik höherer Stufe (HOL: Higher-Order Logic). Durch ihre Ausdrucksstärke ist HOL unter anderem geeignet, um ausführbare Modellierungsnotationen (auch Programmiersprachen, vgl. beispielsweise [10]) zu formalisieren sowie die Eigenschaften der Modellierungsnotation und/oder damit erstellter Modelle formal zu analysieren.

Im weiteren Verlauf dieses Berichts wollen wir die Darstellung von AUTOFOCUS-Modellen in Isabelle/HOL erarbeiten sowie die Äquivalenz der dadurch modellierten AUTOFOCUS-Semantik und der Definition der AUTOFOCUS-Semantik in [12, Abschnitt 4.1.2] sowie im Abschnitt 4.2.1 zeigen.

2 Isabelle/HOL-Formalisierung der Ströme und Stromoperatoren

Die Ausführungssemantik von AUTOFOCUS [7, 11, 1] wurde in [12, Abschnitt 4.1] mithilfe stromverarbeitender Funktionen formal definiert, deren Syntax und Semantik sich auf die Definitionen stützt, die in der Spezifikationsmethode für interaktive verteilte Systeme FOCUS [5] und in der für dienstorientierte Beschreibungen konzipierten Spezifikationsmethode JANUS [4] verwendet werden. Die Kommunikation zwischen AUTOFOCUS-Komponenten wird als zeitsynchrone Nachrichtenströme dargestellt. Ein Nachrichtenstrom stellt die Kommunikationsgeschichte für einen gerichteten Kanal dar.

2.1 Formalisierung der Ströme

Für die Isabelle/HOL-Formalisierung der AUTOFOCUS-Semantik müssen wir zunächst ausgewählte Definitionen aus [12, Abschnitt 3.1.2] in Isabelle/HOL abbilden.

Für eine Trägermenge M bezeichnet

- M^∞ die Menge aller unendlichen Ströme über M
- M^* die Menge aller endlichen Ströme über M

Die Menge aller Ströme über M ist

$$M^\omega \stackrel{\text{def}}{=} M^* \cup M^\infty \tag{1}$$

Die ε-zeitsynchronen Ströme $M^{\varepsilon *}$, $M^{\varepsilon \infty}$, $M^{\varepsilon \omega}$ über M entsprechen Strömen über der Menge $M^\varepsilon = M \cup \{\varepsilon\}$, so dass Nachrichtenströme neben Nachrichten $m \in M$ auch leere Nachrichten ε enthalten dürfen:

datatype *'a message-af = NoMsg | Msg 'a*
syntax (*latex*)
NoMsg :: *'a* (*ε*)
Msg :: *'a* (*Msg*)

In Isabelle/HOL verwenden wir endliche Listen (Datentyp *list*) zur Definition endlicher Ströme und unendliche Listen/Funktionen zur Definition unendlicher Ströme:

types *'a fstream-af = 'a message-af list*
types *'a ilist = nat ⇒ 'a*
types *'a istream-af = 'a message-af ilist*

Da endliche und unendliche Ströme in Isabelle/HOL durch unterschiedliche Datentypen dargestellt werden, müssen im Folgenden separate Stromoperatoren jeweils für endliche und unendliche Ströme definiert werden. Hierbei können wir für endliche Ströme auf eine umfangreiche Bibliothek von Listenoperatoren zurückgreifen (vgl. Theorie *List* [8]).

Ein leerer Strom $\langle\rangle$ entspricht in Isabelle/HOL dem leeren endlichen Strom *[]*. Ein Strom aus n aufeinander folgenden Nachrichten $m_1, \ldots, m_n$ wird als endlicher Strom *[m₁,...,mₙ]* geschrieben.

2.2 Operatoren auf Strömen

Die in [12, Abschnitt 3.1.2] definierten Operatoren werden wie folgt in Isabelle/HOL für endliche und unendliche Ströme dargestellt:

- Die Länge $\mathsf{length}(s)$ eines endlichen Stroms s wird mit dem Listenoperator *length* ermittelt. Weil für unendliche Ströme die Länge stets gleich ∞ ist, wird kein gesonderter Operator für unendliche Ströme codiert.
- Das n-te Element $s.n$ eines Stroms s wird wie folgt ermittelt:
  - Das n-te Elements eines endlichen Stroms wird mithilfe des Listenoperators *!* ermittelt: *s ! n*. Der Zugriff liefert nur für *n < length s* ein definiertes Ergebnis.
  - Für unendliche Ströme liefert die Funktionsanwendung das n-te Element: *s n*.
- Die Konkatenation zweier Ströme $s_1 \frown s_2$ wird wie folgt codiert:
  - Zwei endliche Ströme s_1 und s_2 werden mithilfe des Listenoperators *@* konkateniert: s_1 *@* s_2.
  - Für die Konkatenation eines endlichen und eines unendlichen Stroms wird der Operator $\frown$ definiert:

    constdefs
    i-append :: *'a list ⇒ 'a ilist ⇒ 'a ilist* (**infixr** $\frown$ *65*)
    xs $\frown$ *f* ≡ *λn. if n < length xs then xs ! n else f* (*n − length xs*)

  - Für den Fall, dass der erste Strom s_1 unendlich ist, wird kein Konkatenationsoperator codiert, weil ein unendlicher Strom durch das Anhängen eines beliebigen weiteren Stroms s_2 nicht verändert wird, d.h., $s_1 \frown s_2 = s_1$.

- Das Voranstellen einer Nachricht m einem Strom s wird wie folgt dargestellt
  - Für einen endlichen Strom s wird der Listenoperator # verwendet: *m # s*.
  - Für einen unendlichen Strom s und eine Nachricht m wird der Konkatenationsoperator verwendet: *[m]* $\frown$ *s*.

- Die Schnittoperatoren $s\downarrow_k$ und $s\uparrow_k$ liefern die ersten k Elemente eines Stroms bzw. den restlichen Strom ohne die ersten k Elemente.
  - Für endliche Ströme genügt es, die Listenoperatoren *take* und *drop* syntaktisch anzupassen:

    syntax (*xsymbols*)
    -f-take :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixl** ↓ *100*)
    -f-drop :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixl** ↑ *100*)
    translations
    xs ↓ *n* ⇌ *take n xs*
    xs ↑ *n* ⇌ *drop n xs*
  - Für unendliche Ströme müssen entsprechende Operatoren definiert werden:

    consts
    i-take :: *nat* ⇒ *'a ilist* ⇒ *'a list*
    i-drop :: *nat* ⇒ *'a ilist* ⇒ *'a ilist*
    defs
    i-take-def: *i-take n f* ≡ *map f* [*0..<n*]
    i-drop-def: *i-drop n f* ≡ λ*x*. *f* (*n* + *x*)
    syntax (*xsymbols*)
    -i-take :: *'a ilist* ⇒ *nat* ⇒ *'a list* (**infixl** ⇓ *100*)
    -i-drop :: *'a ilist* ⇒ *nat* ⇒ *'a ilist* (**infixl** ⇑ *100*)
    translations
    f ⇓ *n* ⇌ *i-take n f*
    f ⇑ *n* ⇌ *i-drop n f*

    Um die Operatoren für endliche und unendliche Ströme in Isabelle/HOL syntaktisch unterscheiden zu können, werden für unendliche Ströme Doppelpfeile verwendet.

- Die Operatoren hd, tl, last haben für endliche Ströme gleich benannte Entsprechungen als Listenoperatoren *hd*, *tl*, *last*, wobei die Operatoren *hd* und *last* nur für nicht-leere Ströme ein definiertes Ergebnis liefern.

  Für unendliche Ströme wird die Wirkung dieser Operatoren durch Elementzugriffs- bzw. Schnittoperatoren erreicht:
  - Dem Operator hd entspricht der Zugriff auf das erste Element eines unendlichen Stroms: *s 0*.
  - Dem Operator tl entspricht das Weglassen des ersten Elements eines unendlichen Stroms: *s* ⇑ *1* bzw. äquivalent *s* ⇑ *Suc 0*.
  - Der Operator last hat keine Entsprechung für unendliche Ströme, da ein unendlicher Strom kein letztes Element besitzt.

- Die elementweise Anwendung einer Funktion f auf alle Elemente eines Stroms s wird mit $\mathsf{map}(f,\ s)$ bezeichnet.
  - Für endliche Ströme wird der gleich benannte Listenoperator *map* verwendet: *map f s*.
  - Für unendliche Ströme wird die funktionale Komposition verwendet: *f* ∘ *s*.

3 Isabelle/HOL-Formalisierung der AUTOFOCUS-Semantik

Das Verhalten wird in AUTOFOCUS durch Zustandsübergangsdiagramme (STD [3]) spezifiziert. Ein STD definiert eine Zustandsübergangsfunktion, die aus dem lokalen Zustand der Komponente und der neuen Eingabe den neuen Zustand und die Ausgabe der Komponente berechnet – die Semantik des für die Isabelle/HOL-Darstellung verwendeten STD-Ausschnitts wird im Abschnitt 4.2.1 definiert.

In diesem Bericht betrachten wir herkömmliche AUTOFOCUS-Kommunikationsports und -Kanäle, bei denen die Ausgabe eines Ausgabeports im nächsten Berechnungsschritt an den mit diesem Port verbundenen Eingabeports ankommt. Für die sogenannten Immediate Ports, die die Ausgabe in demselben Berechnungsschritt weiterleiten, müssen bei der Formalisierung einige zusätzliche Probleme gelöst werden, um die Kausalität des Modells zu gewährleisten. Da dieser Ports für die im Projekt betrachtete Fallstudie nach derzeitigem Stand nicht benötigt werden, sehen wir von ihrer Betrachtung ab, und konzentrieren uns auf die herkömmlichen Kommunikationsports.

3.1 Schrittsemantik und Stromverarbeitungssemantik

Wir behandeln nun die Schrittsemantik von AUTOFOCUS (vgl. z.B. [6] sowie Eintaktsemantik in [12, Abschnitt 4.1.2]) und die sich daraus ergebende Stromverarbeitungssemantik, die zu einer Komponente mit einer durch eine Transitionsfunktion gegebenen Schrittsemantik die stromverarbeitende Funktion angibt.

Zunächst beschreiben wir informell den Ablauf eines Berechnungsschritts für eine Komponente. Ein Berechnungsschritt lässt sich in folgende Teilschritte unterteilen (Abbildung 2):

1) **Einlesen der Eingaben**: Die im vorherigen Schritt $t-1$ produzierten Ergebnisse werden übertragen. Falls an einem Ausgabeport keine Nachricht ausgegeben wurde (dies ist auch für $t = 0$ der Fall), wird die leere Nachricht ε als Eingabe übermittelt.

2) **Berechnung**: Alle Komponenten führen ihre Berechnungen durch. Die eingelesene Eingabe und der interne Komponentenzustand werden zur Berechnung der Ausgabe verwendet.

3) **Ausgabe**: Die berechneten Ausgabewerte werden an die Ausgabeports übergeben. Für die Umgebung sind die Ergebnisse erst im nächsten Schritt $t + 1$ sichtbar, als sie im Teilschritt (1) übertragen werden.

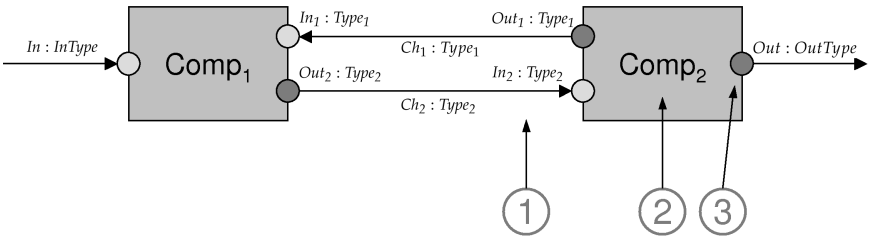

Abbildung 2: Teilschritte eines Berechnungsschritts

Sei C eine Komponente mit Eingabeports

$$\mathcal{IP}_C = \{ip_1, \ldots, ip_{n_{in}}\}$$

und Ausgabeports

$$\mathcal{OP}_C = \{op_1, \ldots, op_{n_{out}}\}$$

Ihre syntaktische Schnittstelle ist dann $(I \rhd O)$ mit

$$I = \mathsf{Type}(ip_1) \times \ldots \times \mathsf{Type}(ip_{n_{in}})$$
$$O = \mathsf{Type}(op_1) \times \ldots \times \mathsf{Type}(op_{n_{out}})$$

Wir bezeichnen mit F_C die stromverarbeitende Funktion, die das Verhalten von C beschreibt. Für jede Eingabehistorie $x \in I^{\varepsilon\omega}$ der Komponente C liefert $F_C(x).t$ ihre Ausgabe am Ende des Berechnungsschritts t.

Die Komponente selbst bestehe aus einem internen Zustand S und der zuletzt produzierten Ausgabe O. Folgende Funktionen werden für eine Komponente C verwendet:

- $\delta_C : I \times C \to C$
  Zustandsübergangsfunktion, die aus dem aktuellen Zustand der Komponente und aktueller Eingabe den nächsten Zustand berechnet, der den lokalen Zustand und die Ausgabe beinhaltet. (Wir verwenden eine zusammengesetzte Darstellung der Komponente als interner Zustand und Ausgabe, um die Formalisierung zu vereinfachen. Durch den Zugriff der Zustandsübergangsfunktion auf die Ausgabe wird die Ausdrucksstärke nicht verändert, denn jede Ausgabe kann von der Komponente zusätzlich in lokalen Variablen gespeichert werden, so dass jede Zustandsübergangsfunktion, die die Ausgabe bei der Berechnung berücksichtigt, durch eine äquivalente Funktion ersetzt werden kann, die entsprechende Ergebniswerte aus lokalen Variablen bezieht und nicht auf die Ausgabe zugreifen muss).

- $\sigma_C : C \to S$
  Funktion zur Extraktion des lokalen Zustands S der Komponente aus dem aktuellen Komponentenzustand C.

- $\rho_C : C \to O$
  Funktion zur Extraktion der berechneten Ausgabewerte O der Komponente aus dem aktuellen Komponentenzustand C.

- $\kappa_C : S \times O \to C$
  Funktion zum Aufbau der Komponente C aus dem lokalen Zustand und den Ausgabewerten. Sie ist das Gegenstück zu den Extraktionsfunktionen σ_C und ρ_C. Es gilt:

$$\begin{aligned} \sigma_C(\kappa_C(s,o)) &= s \\ \rho_C(\kappa_C(s,o)) &= o \\ \kappa_C(\sigma_C(c), \rho_C(c)) &= c \end{aligned}$$

Die Zustandsübergangsfunktion δ_C kann kanonisch zur Funktion $\Delta_C : I^{\varepsilon*} \times C \to C$ erweitert werden, die den Zustand von C nach der Verarbeitung eines (endlichen) Eingabestroms $s \in I^{\varepsilon*}$ berechnet. Wir verwenden eine generische Funktion $\Delta : (I \times C \to C) \times I^{\varepsilon*} \times C \to C$, die mit der Zustandsübergangsfunktion einer Komponente parametriert wird. Δ wird rekursiv über den Aufbau des Eingabestroms definiert:

$$\begin{aligned} \Delta(\delta_C, \langle\rangle, c) &= c \\ \Delta(\delta_C, m\#s, c) &= \Delta(\delta_C, s, \delta_C(m,c)) \end{aligned} \tag{2}$$

Damit liefert $\Delta(\delta_C, s, c)$ den Zustand der Komponente C nach der vollständigen Verarbeitung des Stroms s. Dieser Zustand enthält den lokalen Zustand $\sigma_C(c)$ und die Ausgabe $\rho_C(c)$.

Die Isabelle/HOL-Darstellung der Funktion Δ_C entspricht der obigen Definition. Der Datentyp der Transitionsfunktion δ_C ist

types $('comp, 'input)$ *Comp-Trans-Fun* $=$ *'input* $\Rightarrow$ *'comp* $\Rightarrow$ *'comp*

Die primitiv rekursive Definition von Δ_C ist:

consts *f-Exec-Comp* ::
$('comp, 'input)$ *Comp-Trans-Fun* $\Rightarrow$ *'input list* $\Rightarrow$ *'comp* $\Rightarrow$ *'comp*
primrec
f-Exec-Nil: *f-Exec-Comp trans-fun* [] *c* = *c*
f-Exec-Cons: *f-Exec-Comp trans-fun* $(x\#xs)$ *c* =
f-Exec-Comp trans-fun xs (*trans-fun x c*)

Ähnlich zu Δ definieren wir die Funktion $\Delta^\omega : (I \times C \to C) \times I^{\varepsilon\omega} \times C \to C^\omega$, die die gesamte Geschichte der Verarbeitung eines Stroms durch eine Komponente berechnet, indem sie einen Eingabestrom auf den Strom der Komponentenzustände abbildet, die bei seiner Verarbeitung durchlaufen werden:

$$\begin{array}{lcl} \Delta^\omega(\delta_C,\, \langle\rangle,\, c) & = & \langle\rangle \\ \Delta^\omega(\delta_C,\, m\#s,\, c) & = & \delta_C(m,c)\#\Delta^\omega(\delta_C,\, s,\, \delta_C(m,c)) \end{array} \tag{3}$$

In der Isabelle/HOL-Formalisierung werden einzelne Definitionen für endliche und unendliche Ströme erstellt. Für endliche Ströme wird Δ^ω rekursiv definiert:

consts
f-Exec-Comp-Stream ::
('comp, 'input) *Comp-Trans-Fun* ⇒ 'input *list* ⇒ 'comp ⇒ 'comp *list*
primrec
f-Exec-Stream-Nil: *f-Exec-Comp-Stream trans-fun* [] *c* = []
f-Exec-Stream-Cons: *f-Exec-Comp-Stream trans-fun* (*x* # *xs*) *c* =
(*trans-fun x c*) # (*f-Exec-Comp-Stream trans-fun xs* (*trans-fun x c*))

Die Definition für unendliche Ströme greift auf die Definition für endliche Ströme zurück:

consts *i-Exec-Comp-Stream* ::
('comp, 'input) *Comp-Trans-Fun* ⇒ 'input *ilist* ⇒ 'comp ⇒ 'comp *ilist*
defs *i-Exec-Comp-Stream-def* :
i-Exec-Comp-Stream ≡ λ*trans-fun input c n*.
f-Exec-Comp trans-fun (*input* ⇓ *Suc n*) *c*

$\Delta^\omega(\delta_C,\, s,\, c).n$ stellt damit den Zustand der Komponente C nach der Verarbeitung des Stroms s bis einschließlich zur Position n dar, d.h., des Teilstroms $s\downarrow_{n+1} = \langle s.0, \ldots, s.n\rangle$. In der Isabelle/HOL-Darstellung lauten die Ergebnisse für endliche und unendliche Ströme wie folgt:

lemma *f-Exec-Stream-nth*:
⋀*n c*. *n* < *length xs* ⟹
f-Exec-Comp-Stream trans-fun xs c ! *n* = *f-Exec-Comp trans-fun* (*xs* ↓ *Suc n*) *c*
lemma *i-Exec-Stream-nth*:
i-Exec-Comp-Stream trans-fun input c n = *f-Exec-Comp trans-fun* (*input* ⇓ *Suc n*) *c*

Für einen endlichen Strom $s \in I^{\varepsilon *}$ ist das letzte Element von $\Delta^\omega(\delta_C,\, s,\, c)$ genau der Zustand von C nach der Verarbeitung des gesamten Stroms s:

lemma *f-Exec-eq-f-Exec-Stream-last-if*:
f-Exec-Comp trans-fun xs c = (*if xs* = [] *then c else last* (*f-Exec-Comp-Stream trans-fun xs c*))

Nun lässt sich die Stromverarbeitungsfunktion, die durch die Komponente C gegeben wird (insbesondere durch die Transitionsfunktion *trans-fun* und den Initialzustand *c*) für jede Ausgabeextraktionsfunktion *output-fun* für einzelne oder mehrere Ausgabeports (insbesondere auch für ρ_C, mit welcher sich die durch C definierte Stromverarbeitungsfunktion F_C ergibt) mithilfe von Δ^ω direkt definieren. Die Stromverarbeitungsfunktionen lauten jeweils

- Für endliche Ströme: *map output-fun* (*f-Exec-Comp-Stream trans-fun input c*)
- Für unendliche Ströme: *output-fun* ∘ *i-Exec-Comp-Stream trans-fun input c*

(4)

Nun können einige wichtige Eigenschaften der Zustandsübergangsfunktion Δ, der Zustandsstromfunktion Δ^ω sowie der davon abgeleiteten Ausgabestromspezifikation F in Isabelle/HOL verifiziert werden. Diese Eigenschaften betrachten die Berechnungsströme für ein System bis zu und ab einem beliebigen Zeitpunkt und stellen sicher, dass der endliche Berechnungsstrom vor einem Zeitpunkt zusammen mit dem (endlichen oder unendlichen) Berechnungsstrom nach diesem Zeitpunkt die Berechnung äquivalent zu dem gesamten Berechnungsstrom beschreibt – dies ist eine fundamentale Anforderung an die formale Definition der Semantik, denn sie ermöglicht die Betrachtung und die Analyse von Eigenschaften einzelner Berechnungszustände und von ihnen ausgehend des weiteren Verlaufs einer Berechnung, bei der nicht die gesamte vorhergehende Berechnung, sondern nur der dabei zuletzt erreichte Zustand betrachtet werden kann, was der Annahme entspricht, dass eine Systemberechnung nach jedem Zeitpunkt nur von dem aktuellen Systemzustand und den ab dem aktuellen Zeitpunkt ankommenden Eingaben abhängt.

Die Länge des Zustandsstroms (und damit auch die Länge des Ausgabestroms für jede Ausgabeextraktionsfunktion) ist gleich der Länge des Eingabestroms:

lemma *f-Exec-Stream-length*[*rule-format*]:
$\forall\, c.$ *length* (*f-Exec-Comp-Stream trans-fun xs c*) = *length xs*

Das Ergebnis der Verarbeitung eines konkatenierten Stroms $xs \frown ys$ bzw. $xs \frown input$ für zwei endliche Ströme bzw. einen endlichen und einen unendlichen Strom entspricht der Konkatenation des Ergebnisses für ys bzw. $input$ mit dem Ergebnis für xs, wobei die Verarbeitung des zweiten Stroms beim Komponentenzustand beginnt, der nach der Verarbeitung des ersten Stroms erreicht wurde:

lemma *f-Exec-Stream-append*[*rule-format*]:$\forall\, c.$
f-Exec-Comp-Stream trans-fun (*xs* @ *ys*) *c* =
(*f-Exec-Comp-Stream trans-fun xs c*) @
(*f-Exec-Comp-Stream trans-fun ys* (*f-Exec-Comp trans-fun xs c*))
lemma *i-Exec-Stream-append*:
i-Exec-Comp-Stream trans-fun (*xs* ⌢ *input*) *c* =
f-Exec-Comp-Stream trans-fun xs c ⌢
i-Exec-Comp-Stream trans-fun input (*f-Exec-Comp trans-fun xs c*)

Die Ausführung eines Rechenschritts ist damit ein Spezialfall der Konkatenation, bei dem der zweite Eingabestrom genau ein Element enthält:

lemma *f-Exec-Stream-snoc*:
f-Exec-Comp-Stream trans-fun (*xs* @ [*x*]) *c* =
f-Exec-Comp-Stream trans-fun xs c @
[*trans-fun x* (*f-Exec-Comp trans-fun xs c*)]

Analog liefert die Zustandsübergangsfunktion Δ für die Konkatenation $xs \frown ys$ zweier Ströme den Zustand, der nach der Verarbeitung von ys erreicht wird, wobei die Ausführung beim Zustand beginnt, der nach der Verarbeitung von xs erreicht wurde:

corollary *f-Exec-append*[*rule-format*]:
f-Exec-Comp trans-fun (*xs* @ *ys*) *c* =
f-Exec-Comp trans-fun ys (*f-Exec-Comp trans-fun xs c*)

Die Ausführung eines Rechenschritt entspricht wiederum der Konkatenation mit einem Strom, der genau ein Element enthält:

lemma *f-Exec-snoc*:
f-Exec-Comp trans-fun (*xs* @ [*x*]) *c* =
trans-fun x (*f-Exec-Comp trans-fun xs c*)

Das Zusammenspiel der Schnittoperatoren auf Strömen mit den Stromverarbeitungsfunktionen Δ^ω und F ist ebenfalls sehr direkt. Das Abschneiden des Ergebnisstroms ab dem n-ten Element liefert dasselbe Ergebnis, wie die Verarbeitung des abgeschnittenen Eingabestroms:

theorem *f-Exec-Stream-take*:
(*f-Exec-Comp-Stream trans-fun xs c*) ↓ *n* =
f-Exec-Comp-Stream trans-fun (*xs* ↓ *n*) *c*
theorem *i-Exec-Stream-take*:
(*i-Exec-Comp-Stream trans-fun input c*) ⇓ *n* =
f-Exec-Comp-Stream trans-fun (*input* ⇓ *n*) *c*

Das Weglassen der ersten n Elemente des Ergebnisstroms entspricht der Verarbeitung des Eingabestroms, dessen erste n Elemente weggelassen wurden, wobei die Ausführung im Zustand beginnt, der nach der Verarbeitung der ersten n Elemente erreicht wird:

theorem *f-Exec-Stream-drop*:
(*f-Exec-Comp-Stream trans-fun xs c*) ↑ *n* =
f-Exec-Comp-Stream trans-fun (*xs* ↑ *n*)
(*f-Exec-Comp trans-fun* (*xs* ↓ *n*) *c*)
theorem *i-Exec-Stream-drop*:
(*i-Exec-Comp-Stream trans-fun input c*) ⇑ *n* =

i-Exec-Comp-Stream trans-fun $(input \Uparrow n)$
$(f$*-Exec-Comp trans-fun* $(input \Downarrow n)\ c)$

Somit können für die Betrachtung eines Berechnungsablaufs vor bzw. nach einem bestimmten Zeitpunkt die Stromschnittoperatoren auf dem Ergebnisstrom oder dem Eingabestrom genutzt werden.

Diese und weitere Eigenschaften der Stromverarbeitungsfunktionen, die bei der Formalisierung der AUTOFOCUS-Semantik verwendet werden, finden sich in [12, Anhang A.1].

3.2 Verhaltensspezifikation durch Zustandsautomaten

Wir befassen uns zunächst mit der Semantik von Komponenten, deren Verhalten durch ein Zustandsübergangsdiagramm (STD) spezifiziert wird. Diese Komponenten bilden die Blätter des Strukturbaums eines AUTOFOCUS-Modells – die Transitionsfunktion wird direkt durch die Transitionsfunktion des zugeordneten Zustandsautomaten definiert.

3.2.1 SSD

Im Folgenden wird die Isabelle/HOL-Darstellung des SSDs einer Komponente behandelt, deren Verhalten durch ein STD spezifiziert wird.

Die Eingabe/Ausgabeschnittstellen werden durch *record*-Datentypen codiert, wobei für jeden Eingabe/Ausgabeport <*Name-i*> ein *record*-Element mit entsprechendem Datentyp erstellt wird:

record <*ComponentName*>*-SSD-InputPorts-type* $=$
<*ComponentName*>*-InPort-*<*Name-1*> :: <*type*> *message-af*
<*ComponentName*>*-InPort-*<*Name-2*> :: <*type*> *message-af*
...
record <*ComponentName*>*-SSD-OutputPorts-type* $=$
<*ComponentName*>*-OutPort-*<*Name-1*> :: <*type*> *message-af*
<*ComponentName*>*-OutPort-*<*Name-2*> :: <*type*> *message-af*
...

Für die Eingabe/Ausgabedatentypen werden *is-NoMsg*-Prädikate erstellt, die für eine Eingabe/Ausgabe ermitteln, ob alle ein/-ausgehenden Werte leer sind. Das *is-NoMsg*-Prädikat für die Eingabe sieht wie folgt aus:

defs (**overloaded**)
is-NoMsg-<*ComponentName*>*-SSD-InputPorts-type* :
is-NoMsg $(m::$<*ComponentName*>*-SSD-InputPorts-type*$) \equiv$
<*ComponentName*>*-InPort-*<*Name-1*> $m = \varepsilon \wedge$
<*ComponentName*>*-InPort-*<*Name-2*> $m = \varepsilon \wedge$
...

Ein *is-NoMsg*-Prädikat für die Ausgabeschnittstelle wird analog definiert.

In der Isabelle/HOL-Darstellung werden Eingabeströme von Komponenten stets als Nachrichtenströme codiert – besteht die Eingabe aus Nachrichten für mehrere Ports, so muss der die Eingabe darstellende Record als Nachricht verpackt werden. Das Entpacken der Eingabe für die Komponente übernimmt die Funktion *the-*<*ComponentName*>*-SSD-InputPorts*:

consts *the-*<*ComponentName*>*-SSD-InputPorts* ::
<*ComponentName*>*-SSD-InputPorts-type message-af* $\Rightarrow$ <*ComponentName*>*-SSD-InputPorts-type*
primrec
the-<*ComponentName*>*-SSD-InputPorts* $\varepsilon = (\!|$
<*ComponentName*>*-InPort-*<*Name-1*> $= \varepsilon$,
<*ComponentName*>*-InPort-*<*Name-2*> $= \varepsilon$,
$\ldots |\!)$
the-<*ComponentName*>*-SSD-InputPorts* $(Msg\ inputs) = inputs$

Hierbei entspricht eine leere Eingabe einem Eingaberecord, in dem sämtliche Nachrichten leer sind. Dadurch wird die punktweise Erweiterung der leeren Nachricht ε auf Nachrichtenrecords modelliert, die in

der AUTOFOCUS-Semantik in [12, Abschnitt 3.1.2] für Nachrichtentupel definiert wurde:

$$\forall m \in (M_1 \times \ldots \times M_n)^{\varepsilon} : \quad m = \varepsilon \;\Leftrightarrow\; \forall i \in [1 \ldots n] : \Pi_i(m) = \varepsilon \tag{5}$$

Der dabei verwendete Projektionsoperator dient zum Zugriff auf einzelne Elemente eines Tupels:

$$\Pi_j(e_1, \ldots, e_n) = e_j$$

Die lokalen Variablen einer Komponente werden ebenfalls in einem Record zusammengefasst:

record *<ComponentName>-SSD-LocalVariables-type* =
Digits2-LocVar-<Name-1> :: *<type>*
Digits2-LocVar-<Name-2> :: *<type>*
...

Das SSD enthält in der Isabelle/HOL-Darstellung die syntaktische Ausgabeschnittstelle sowie lokale Variable der Komponente:

record *<ComponentName>-SSD* =
<ComponentName>-SSD-OutPorts :: *<ComponentName>-SSD-OutputPorts-type*
<ComponentName>-SSD-LocVars :: *<ComponentName>-SSD-LocalVariables-type*

Die Eingabeschnittstelle ist nicht Teil des SSDs, da das SSD die Berechnungsergebnisse (interner Komponentenzustand sowie Ausgabe) darstellt.

Da der Zugriff auf Ausgabewerte sowie lokale Variable der Anwendung mehrerer Selektorfunktionen für die entsprechenden Records in dem SSD-Record bedarf, werden diese durch eigene Zugriffsfunktionen abgekürzt. Die Zugriffsfunktion für einen Ausgabeport *<Name-i>* wird wie folgt konstruiert:

consts
get-<ComponentName>-OutPort-<Name-i> :: *<ComponentName>-SSD* ⇒ *<type> message-af*
defs
get-<ComponentName>-OutPort-<Name-i>-def :
get-<ComponentName>-OutPort-<Name-i> ssd ≡
<ComponentName>-OutPort-<Name-i> (*<ComponentName>-SSD-OutPorts ssd*)
consts
set-<ComponentName>-OutPort-<Name-i> ::
<type> message-af ⇒ *<ComponentName>-SSD* ⇒ *<ComponentName>-SSD*
defs
set-<ComponentName>-OutPort-<Name-i>-def :
set-<ComponentName>-OutPort-<Name-i> m ssd ≡
ssd (| *<ComponentName>-SSD-OutPorts* :=
(*<ComponentName>-SSD-OutPorts ssd*) (| *<ComponentName>-OutPort-<Name-i>* := *m* |) |)

Die Zugriffsfunktionen für lokale Variable werden analog erzeugt:

consts
get-<ComponentName>-LocVar-<Name-i> :: *<ComponentName>-SSD* ⇒ *<type>*
defs
get-<ComponentName>-LocVar-<Name-i>-def :
get-<ComponentName>-LocVar-<Name-i> ssd ≡
<ComponentName>-LocVar-<Name-i> (*<ComponentName>-SSD-LocVars ssd*)

consts
set-<ComponentName>-LocVar-<Name-i> ::
<type> ⇒ *<ComponentName>-SSD* ⇒ *<ComponentName>-SSD*
defs
set-<ComponentName>-LocVar-<Name-i>-def :
set-<ComponentName>-LocVar-<Name-i> x ssd ≡
ssd (| *<ComponentName>-SSD-LocVars* :=
(*<ComponentName>-SSD-LocVars ssd*) (| *<ComponentName>-LocVar-<Name-i>* := *x* |) |)

Die Transitionssemantik von AUTOFOCUS schreibt das Leeren der Ausgabeports vor einer neuen Ausgabe vor. Diese Aufgabe wird von der Funktion *flushOutputPorts-<ComponentName>-SSD* erfüllt:

constdefs
flushOutputPorts-<ComponentName>-SSD :: *<ComponentName>-SSD* $\Rightarrow$ *<ComponentName>-SSD*
flushOutputPorts-<ComponentName>-SSD ssd $\equiv$
ssd (| *<ComponentName>-SSD-OutPorts* := (|
<ComponentName>-OutPort-<Name-1> $= \varepsilon$,
<ComponentName>-OutPort-<Name-2> $= \varepsilon$,
... |) |)

Schließlich definieren wir den Initialzustand des SSDs. In dem Initialzustand sind alle Ausgaben leer, die internen Variablen dürfen einen beliebigen aber festen, durch den Komponentenentwickler zugewiesenen Initialwert haben.

consts
<ComponentName>-SSD-InitValue :: *<ComponentName>-SSD*

defs
<ComponentName>-SSD-InitValue-def : *<ComponentName>-SSD-InitValue* $\equiv$
(| *<ComponentName>-SSD-OutPorts* $=$ (|
<ComponentName>-OutPort-<Name-1> $= \varepsilon$,
<ComponentName>-OutPort-<Name-2> $= \varepsilon$,
... |),
<ComponentName>-SSD-LocVars $=$ (|
<ComponentName>-LocVar-<Name-1> $=$ *<initial value 1>*,
<ComponentName>-LocVar-<Name-2> $=$ *<initial value 2>*,
... |)
|)

3.2.2 STD

Dir Zustandsübergangsfunktion einer Komponente mit einem Zustandsautomaten wird durch das entsprechende Zustandsübergangsdiagramm (STD) definiert. Eine ausführliche Beschreibung der Zustandsübergangsdiagramme liefert [3], weitere Erläuterungen zu STDs in AUTOFOCUS finden sich in z.B. in [11, 6].

Ein Zustandsübergangsdiagramm besteht aus einer endlichen nicht-leeren Menge von *Kontrollzuständen* sowie einer beliebigen Anzahl von *Transitionen* zwischen Zuständen. Jede Transition hat fünf Bestandteile. Die ersten zwei stellen die Vorbedingung dar, bei deren Erfüllung die Transition schalten kann:

- Das *Eingabemuster* gibt an, welche Ports leer und welche nicht-leer sein müssen. Ferner werden den eingegebenen Portwerten temporäre lokale Bezeichner zugewiesen, die in der Transition verwendet werden können.
- Die *Vorbedingung* gibt ein logisches Prädikat an, bei dessen Erfüllung die Transition schalten kann.

Die letzten drei Bestandteile definieren die Auswirkungen der Transitionsausführung:

- Das *Ausgabemuster* definiert die Ausgaben am Ende der Transition.
- Die *Nachbedingung* definiert den neuen internen Zustand (Werte lokaler Variablen) am Ende der Transition.
- Der *Zielzustand* gibt an, in welchen Kontrollzustand sich der Automat nach der Ausführung der Transition befindet.

Die Kontrollzustände des Automaten werden durch einen Aufzählungsdatentyp dargestellt (der Name des Automaten darf dabei mit dem Komponentennamen übereinstimmen):

datatype
<AutomatonName>-STD-State-type =
<AutomatonName>-<StateName-1> |
<AutomatonName>-<StateName-2> |
...

Der Initialzustand des STD, der Teil des Initialzustands der Komponente sein wird:

constdefs
<AutomatonName>-STD-State-InitValue :: *<AutomatonName>-STD-State-type*
<AutomatonName>-STD-State-InitValue ≡ *<AutomatonName>-<InitStateName>*

Datentyp der Zustandsübergangsfunktion des STDs. Diesen Datentyp haben sowohl Funktionen, die einzelne Zustandsübergänge darstellen, als auch die Haupttransitionsfunktion:

types *<AutomatonName>-STD-TransitionFunction-type* =
<AutomatonName>-STD-State-type ⇒
<ComponentName>-SSD-InputPorts-type ⇒ *<ComponentName>-SSD* ⇒
(*<AutomatonName>-STD-State-type* × *<ComponentName>-SSD*)

Die Leerlauftransition. Diese Transitionsfunktion wird ausgeführt, wenn für keine Transition des Automaten in dem aktuellen Kontrollzustand die Vorbedingungen erfüllt sind:

consts
<AutomatonName>-STD-NoTransition :: *<AutomatonName>-STD-TransitionFunction-type*
defs
<AutomatonName>-STD-NoTransition-def :
<AutomatonName>-STD-NoTransition state input ssd ≡ (*state*, *ssd*)

Für jede Transition aus einem Zustand i wird jeweils eine Transitionsfunktion erstellt, welche die Aktionen der Transition ausführt (die Vorbedingungen werden von ihr selbst nicht überprüft, sondern in der weiter unten definierten Haupttransitionsfunktion):

consts
<AutomatonName>-STD-Transition-<StateName-i>-<StateName-j-1>-1 ::
<AutomatonName>-STD-TransitionFunction-type
<AutomatonName>-STD-Transition-<StateName-i>-<StateName-j-2>-2 ::
<AutomatonName>-STD-TransitionFunction-type
...
defs
<AutomatonName>-STD-Transition-<StateName-i>-<StateName-j-k>-k-def :
<AutomatonName>-STD-Transition-<StateName-i>-<StateName-j-k> state input ssd ≡ (
<AutomatonName>-<StateName-j-k>,
set-<ComponentName>-OutPort-<Name-1> (*Msg <Computed result>*) (
set-<ComponentName>-OutPort-<Name-2> (*Msg <Computed result>*) (
...
set-<ComponentName>-LocVar-<Name-1> <Computed result> (
set-<ComponentName>-LocVar-<Name-2> <Computed result> (
...
set-<ComponentName>-LocVar-<Name-NVar> <Computed result> ssd)...)
)

Für Ports, an die keine neue Ausgabe geschrieben wird, muss keine leere Nachricht explizit ausgegeben werden, weil alle Ports in jedem Berechnungsschritt vor der Berechnung durch den Aufruf von *flush-OutputPorts-<ComponentName>-SSD* geleert werden – es genügt daher, für diese Ports p_l die Zuweisung eines Ausgabewerts durch den Aufruf von *set-<ComponentName>-OutPort-<Name-l>* in der Transitionsfunktion wegzulassen. Ebenfalls können für lokale Variablen, für die kein neuer Wert berechnet wurde, die entsprechenden Zuweisungen weggelassen werden.

Für die Haupttransitionsfunktion wird eine Hilfsfunktion definiert, die ausgehend von dem Komponentenzustand vor dem Berechnungsschritt und der neuen Eingabe eine Zustandstransition auswählt, deren Vorbedingungen erfüllt sind. Die Funktion ist wie folgt aufgebaut:

- Für jeden Zustand <*AutomatonName*>-<*StateName-i*> wird eine entsprechende Zeile definiert.
- Für jede Transition k zu einem Zustand <*AutomatonName*>-<*StateName-j-k*> wird eine entsprechende *if*-Anweisung erstellt.
- Aus dem Eingabemuster einer Transition k wird wie folgt eine entsprechende Vorbedingung <Input pattern i-k> erstellt:
  - Kommt ein Port p in dem Eingabemuster als $p?x$ vor, so ist eine Eingabe gefordert und es wird die Vorbedingung $p \neq \varepsilon$ erstellt.
  - Kommt ein Port p in dem Eingabemuster als $p?$ vor, so darf keine nicht-leere Eingabe eingehen und es wird die Vorbedingung $p = \varepsilon$ erstellt.
  - Kommt ein Port p in dem Eingabemuster nicht vor, so wird für ihn keine Vorbedingung erstellt.
- Die eigentliche Vorbedingung <Precondition i-k> wird direkt aus der Vorbedingung der Transition erzeugt – dabei müssen lediglich Anweisungen der funktionalen Sprache QuestF in ML übersetzt und gegebenenfalls Portnamen durch die im Eingabemuster definierten lokalen Bezeichner ersetzt werden. Die Festlegung einer genauen Übersetzungsvorschrift, die vorwiegend eine syntaktische Transformation darstellen sollte, ist Teil der geplanten Implementierung eines Isabelle/HOL-Codegenerators für AUTOFOCUS.

Die Isabelle/HOL-Formulierung der Transitionsauswahlfunktion sieht wie folgt aus:

consts
<AutomatonName>-STD-TransitionFunction-selectTransition ::
<AutomatonName>-STD-State-type ⇒
<AutomatonName>-SSD-InputPorts-type ⇒ *<AutomatonName>-SSD* ⇒
<AutomatonName>-STD-TransitionFunction-type
primrec
<AutomatonName>-STD-TransitionFunction-selectTransition
<AutomatonName>-<StateName-i> input ssd = (
if (<Input pattern i-1> input ∧ *<Precondition i-1> input ssd)*
then <AutomatonName>-STD-Transition-<StateName-i>-<StateName-j-1>-1
else (if (<Input pattern i-2> input ∧ *<Precondition i-2> input ssd)*
then <AutomatonName>-STD-Transition-<StateName-i>-<StateName-j-2>-2
...
else <AutomatonName>-STD-NoTransition)...)

Durch den Aufbau der Transitionsauswahlfunktion ist die Auswahl deterministisch, d.h., es wird die erste Transition ausgewählt, deren Vorbedingungen erfüllt sind (vgl. Abschnitt 4.2.1). Die Reihenfolge der Transitionen in den *if*-Anweisungen muss einer beliebigen aber festen Ordnung entsprechen, gemäß welcher auch in AUTOFOCUS die Transition ausgewählt wird, wenn die Vorbedingung mehrerer Transitionen erfüllt sind.

Die Haupttransitionsfunktion kann nun als Übergabe des Zustands und der aktuellen Eingabe an eine durch die obige Auswahlfunktion ausgewählte Transitionsfunktion definiert werden:

consts
<AutomatonName>-STD-TransitionFunction :: *<AutomatonName>-STD-TransitionFunction-type*
defs
<AutomatonName>-STD-TransitionFunction-def :
<AutomatonName>-STD-TransitionFunction state input ssd ≡
(*<AutomatonName>-STD-TransitionFunction-selectTransition state input ssd*) *state input ssd*

3.2.3 Komponente

Nachdem die Isabelle/HOL-Darstellungen für das SSD und das STD erzeugt sind, können die gesamte Komponente und ihre Transitionsfunktion kanonisch definiert werden.

Die Komponente umfasst das SSD und den Automatenkontrollzustand:

record *<ComponentName>-Component* =
<ComponentName>-Component-SSD :: *<ComponentName>-SSD*
<ComponentName>-Component-STD-State :: *<AutomatonName>-STD-State-type*

Für Komponententransitionsfunktion wird eine Hilfsfunktion definiert:

constdefs
set-<ComponentName>-Comp-State-and-SSD ::
<AutomatonName>-STD-State-type × *<ComponentName>-SSD* ⇒
<ComponentName>-Component ⇒ *<ComponentName>-Component*
set-<ComponentName>-Comp-State-and-SSD state-ssd c ≡ (|
<ComponentName>-Component-SSD = *snd state-ssd*,
<ComponentName>-Component-STD-State = *fst state-ssd* |)

Nun kann die Transitionsfunktion der Komponente definiert werden. Die Komponententransitionsfunktion leert die Ausgabeports im SSD, und reicht das SSD, den Kontrollzustand sowie die neue Eingabe an die Transitionsfunktion des STDs. Die Berechnungsergebnisse werden dann in den Komponentenzustand übertragen:

consts
<ComponentName>-Comp-TransitionFunction ::
<ComponentName>-SSD-InputPorts-type message-af ⇒
<ComponentName>-Component ⇒ *<ComponentName>-Component*
defs
<ComponentName>-Comp-TransitionFunction-def :
<ComponentName>-Comp-TransitionFunction input c ≡
set-<ComponentName>-Comp-State-and-SSD
(∗ *The new SSD and new STD State* ∗)
(*<AutomatonName>-STD-TransitionFunction*
(*<ComponentName>-Component-STD-State c*)
(*the-<ComponentName>-SSD-InputPorts input*)
(∗ *The old output port values are flushed at the beginning of the processing step* ∗)
(*flushOutputPorts-<ComponentName>-SSD* (*<ComponentName>-Component-SSD c*)))
(∗ *The component to be altered* ∗)
c

3.3 Kompositions- und Kommunikationssemantik

Wir behandeln nun die Isabelle/HOL-Darstellung zusammengesetzter Komponenten, deren Semantik in [12, Abschnitt 4.1, Teilabschnitt Komposition] formalisiert wurde.

Die Semantik einer Komponente wird durch ihre Zustandsübergangsfunktion definiert. Die Semantik eines Modells $\mathcal{M}$ wird durch die Zustandsübergangsfunktion ihrer Wurzelkomponenten $\mathcal{RC}_{\mathcal{M}}$ definiert. Das Modell wird nach außen durch die Schnittstelle $(I_{\mathcal{RC}_{\mathcal{M}}} \triangleright O_{\mathcal{RC}_{\mathcal{M}}})$ der Wurzelkomponente repräsentiert. Ihre Zustandsübergangsfunktion $\delta_{\mathcal{RC}_{\mathcal{M}}}$ wird entlang des Strukturbaums des Modells $\mathcal{M}$ aufgebaut.

Für ein Blatt C des Strukturbaums wird die Spezifikation durch einen Zustandsübergangsautomaten definiert – die Darstellung seiner Zustandsübergangsfunktion δ_C in Isabelle/HOL wird im vorherigen Abschnitt 3.2 behandelt.

Ein Knoten C des Strukturbaums ist eine hierarchisch aufgebaute Komponente, deren Verhalten durch die Zustandsübergangsfunktionen ihrer Teilkomponenten $\delta_{C_1}, \ldots, \delta_{C_n}$ und die Kommunikationsstruktur $\mathcal{CH}_C, \mathcal{IA}_C, \mathcal{OA}_C$ definiert wird.

Sei C eine Komponente, die aus Teilkomponenten $C_1, \ldots, C_n$ besteht. Sie soll über die syntaktische

Schnittstelle $(I \triangleright O)$ mit Eingabeports $\mathcal{IP}_C = \{ip_1, \ldots, ip_{n_{in}}\}$ und Ausgabeports $\mathcal{OP}_C = \{op_1, \ldots, op_{n_{out}}\}$ verfügen. Jede Teilkomponente C_i soll analog über die syntaktische Schnittstelle $(I_i \triangleright O_i)$ mit $\mathcal{IP}_{C_i} = \{ip_{i,1}, \ldots, ip_{i,n_{in,i}}\}$ und $\mathcal{OP}_{C_i} = \{op_{i,1}, \ldots, op_{i,n_{out,i}}\}$ verfügen.

Die Kommunikationsstruktur innerhalb der Komponente C wird durch die Menge ihrer Kommunikationskanäle $\mathcal{CH}_C$ definiert, die Ausgabeports aus der Menge $\bigcup_{i=1}^{n} \mathcal{OP}_{C_i}$ mit den Eingabeports aus der Menge $\bigcup_{i=1}^{n} \mathcal{IP}_{C_i}$ verbinden, sowie durch die Anbindung der Teilkomponenten an die externe Schnittstelle der Komponente C über die Zuordnungen/Assoziationen $\mathcal{IA}_C$ und $\mathcal{OA}_C$ der extern sichtbaren Eingabeports $\mathcal{IP}_C$ und Ausgabeport $\mathcal{OP}_C$ zu den Ports der Teilkomponenten. Dabei dürfen Eingabeports der Teilkomponenten entweder höchstens einen eingehenden Kanal besitzen oder höchstens einem Eingabeport von C zugeordnet sein, damit der Eingabewert eindeutig ist. Ausgabeport dürfen dagegen sowohl mehrere ausgehende Kanäle besitzen, als auch mehreren Ausgabeports von C zugeordnet sein, um ihre Werte an mehrere Empfänger auszugeben. Die einzige Einschränkung für Ausgabeports von C ist, dass einem Ausgabeport $p \in \mathcal{OP}_C$ höchstens ein Ausgabeport einer Teilkomponente zugeordnet werden darf, damit der Ausgabewert an p eindeutig ist.

Wir befassen uns nun mit der Semantik einer hierarchisch aufgebauten Komponente. Der lokale Zustand S einer hierarchisch aufgebauten Komponente C enthält Informationen über die Zustände ihrer Teilkomponenten $C_1, \ldots, C_n$, die zur Berechnung des nächsten Zustands einschließlich der Ausgabe aus dem letzten Zustand und der aktuellen Eingabe notwendig sind:

record *<ComponentName>-SSD-Subcomponents-type* =
<ComponentName>-Subcomponent--<Name1> :: *<ComponentType>*
<ComponentName>-Subcomponent--<Name2> :: *<ComponentType>*
...
<ComponentName>-Subcomponent--<NameN> :: *<ComponentType>*

Der Gesamtzustand von C setzt sich damit aus den Zuständen der Teilkomponenten und der Ausgabe zusammen:

record *<ComponentName>-SSD* =
<ComponentName>-SSD-Subcomponents :: *<ComponentName>-SSD-Subcomponents-type*
<ComponentName>-SSD-OutPorts :: *<ComponentName>-SSD-OutputPorts-type*
record *<ComponentName>-Component* =
<ComponentName>-Component-SSD :: *<ComponentName>-SSD*

Anders als bei Komponenten mit Verhaltensverfeinerung durch STD (vgl. Abschnitt 3.2), hat der Gesamtzustand *<ComponentName>-Component* einer zusammengesetzten Komponente keine weiteren Bestandteile außer des SSD-Zustands *<ComponentName>-SSD*.

3.3.1 Kommunikationssemantik

Um die Zustandsübergangsfunktion δ_C zu spezifizieren, müssen wir die Kommunikationssemantik für die interne Struktur hierarchisch aufgebauter Komponenten definieren. Das wird bewerkstelligt, indem für jeden Port einer Teilkomponentenschnittstelle festgelegt wird, wie sich sein Wert aus dem Komponentenzustand nach dem letzten Berechnungsschritt und der aktuellen Komponenteneingabe ergibt.

Zunächst behandeln wir die Ausgabeports, welche die Ergebnisse der Berechnungen einer Komponente nach außen kommunizieren. Entsprechend der Kommunikationsstruktur der Komponente C wird eine Ausgabeextraktionsfunktion definiert, welche die Ausgabe von C aus ihrem internen Zustand und mithin aus den Ausgaben ihrer Teilkomponenten erzeugt. Für einen Ausgabeport p_k der Komponente C sind zwei Fälle zu unterscheiden:

1) Der Port p_k ist dem Ausgabeport $p_{i,j}$ einer Teilkomponente C_i zugeordnet. In diesem Fall entspricht sein Wert dem Berechnungsergebnis an $p_{i,j}$ am Ende des Berechnungsschritts. Die entsprechende Ausgabeextraktionsfunktion ist:

   <ComponentName>-OutVal-k =
   get-<SubcomponentName-i>-OutPort-j ∘ *<ComponentName>-Subcomponent--<Name-i>*

2) Der Port p_k ist frei, d.h., er ist keinem Ausgabeport einer Teilkomponente von C zugeordnet. In diesem Fall ist seine Ausgabe stets leer:

<ComponentName>-OutVal-k $= \lambda x.\, \varepsilon$

Die entsprechende Isabelle/HOL-Darstellung ist:

consts *<ComponentName>-make-output* ::
<ComponentName>-SSD-Subcomponents-type $\Rightarrow$ $(*$ *Subcomponents* $*)$
<ComponentName>-SSD-OutputPorts-type
defs *<ComponentName>-make-output-def* :
<ComponentName>-make-output subcomps $\equiv$
$(\!|$
<ComponentName>-OutPort-1 $=$ *<ComponentName>-OutVal-1 subcomps*,
$\ldots$,
<ComponentName>-OutPort-N-Out $=$ *<ComponentName>-OutVal-N-Out subcomps*
$|\!)$

Nun definieren wir die Kommunikationssemantik der Eingabeports, indem für jede Teilkomponente C_i eine Funktion definiert wird, welche die Eingabe aus dem letzten Zustand der Komponente C und ihrer aktuellen Eingabe entsprechend der Kommunikationsstruktur der Komponente C erzeugt. Für einen Eingabeport p_k einer Teilkomponente C_i sind drei Fälle zu unterscheiden:

1) Der Port p_k ist mit einem Kanal ch verbunden: $p = \mathsf{DestinationPort}(ch)$. Sei $p_l = \mathsf{SourcePort}(ch)$ der Ausgabeport einer Teilkomponente C_j, mit dem p_k durch ch verbunden ist. Die Berechnungsergebnisse eines Berechnungsschritts t werden am Beginn des nächsten Schritts $t + 1$ übertragen und stehen für die Berechnungen in diesem Schritt zur Verfügung (am Beginn der Systemausführung wurden noch keine Ergebnisse ausgegeben, so dass zum Zeitpunkt $t = 0$ leere Nachrichten übertragen werden):

   <SubcomponentName-i>-InPort-<Port-k> $=$
   get-<SubcomponentName-j>-OutPort-<Port-l> (
   get-<ComponentName>-Subcomponent--<Name-j> c)

2) Der Port p_k ist mit einem Eingabeport p_l der Oberkomponente C von C_i assoziiert: $(p_k, p_l) \in \mathcal{IA}_C$. Wie bereits im Fall eines assoziierten Ausgabeports, ist der an p_k anliegende Wert stets gleich dem Wert an p_l:

   <SubcomponentName-i>-InPort-<Port-k> $=$
   <ComponentName>-InPort-<Port-l> (the-<ComponentName>-SSD-InputPorts input)

3) Der Port p ist frei, d.h., er ist weder mit einem anderen Port durch einen Kanal verbunden, noch mit einem Eingabeport der Schnittstelle von C assoziiert. An einem solchen Eingabeport kommen keine Eingaben an, so dass er stets leer ist:

   <SubcomponentName>-InPort-<Port-k> $= \varepsilon$

Die sich ergebende Funktion zur Konstruktion der Eingabe sieht in Isabelle/HOL wie folgt aus:

consts *<ComponentName>-make-input--<SubcomponentName-i>* ::
<ComponentName>-SSD-InputPorts-type message-af $\Rightarrow$ $(*$ *External input* $*)$
<ComponentName>-Component $\Rightarrow$ $(*$ *Internal state* $*)$
<SubomponentName-i>-SSD-InputPorts-type message-af
defs *<ComponentName>-make-input--<SubcomponentName-i>-def* :
<ComponentName>-make-input--<SubcomponentName-i> input c $\equiv$
let input $=$ $(\!|$
<SubcomponentName-i>-InPort-<Name-j> $=$
<Source port value from input or c according to communication structure>,
$\ldots$ $|\!)$
in
if is-NoMsg input then ε *else Msg input*

3.3.2 Zustandsübergangsfunktion

Wir definieren nun die Zustandsübergangsfunktion einer aus mehreren Teilkomponenten zusammengesetzten Komponente C. Die Zustandsübergangsfunktion δ_C führt einen Berechnungsschritt aus, indem alle Teilkomponenten einen Berechnungsschritt mit den Eingaben ausführen, die sich aus den Eingabewerten für C und den Ausgaben der Teilkomponenten aus dem vorherigen Berechnungsschritt zusammen mit der Kommunikationsstruktur von C ergeben. Dabei wird vorausgesetzt, dass die Ausgabeports einer jeden Komponente am Beginn des Systemlaufs leer sind, d.h., alle Ausgabeports den Initialwert ε vor dem Systemstart haben – zum Zeitpunkt der Datenübertragung (erster Teilschritt) des Berechnungsschritts für $t = 0$ gilt deshalb: $\forall C \in \mathcal{C}_M : \ \rho_C(C) = \varepsilon$. Somit ergibt sich die Zustandsübergangsfunktion δ_C einer zusammengesetzten Komponente C mit Teilkomponenten $C_1, \ldots, C_n$ aus den Zustandsübergangsfunktionen der Teilkomponenten und der Kommunikationsstruktur der Komponente C.

Wir bauen die Zustandsübergangsfunktion in einer *let*-Anweisung entsprechend den drei Teilschritten eines Berechnungsschritts zusammen (vgl. Abschnitt 3.1). Die Zustandsübergangsfunktion berechnet für einen Komponentenzustand und eine neue externe Eingabe einen neuen Komponentenzustand:

consts
<ComponentName>-Comp-TransitionFunction ::
<ComponentName>-SSD-InputPorts-type message-af $\Rightarrow$ *<ComponentName>-Component* $\Rightarrow$
<ComponentName>-Component

Die Teilschritte sind die folgenden:

1) Zunächst werden die neuen Eingaben aus den Ausgaben im letzten Berechnungszustand und den neuen externen Eingaben konstruiert. Hierzu werden die oben definierten *make-input*-Funktionen genutzt:

<SubcomponentName-i>-input $=$
<ComponentName>-make-input--<SubcomponentName-i> input c;

2) Anschließend werden die Berechnungen durch die Teilkomponenten durchgeführt:

<SubcomponentName-i>-new $=$
<SubcomponentName-i>-Comp-TransitionFunction
<SubcomponentName-i>-input
(*get-<ComponentName>-Subcomponent--<SubcomponentName-i> c*)

3) Zuletzt wird der neue Komponentenzustand aus den neuen Komponentenzuständen der Teilkomponenten konstruiert. Nach der Zusammensetzung der neuen Teilkomponentenstruktur

subcomps-new $= (\!|$
<ComponentName>-Subcomponent--<SubcomponentName1> $=$ *<SubcomponentName1>-new*,
<ComponentName>-Subcomponent--<SubcomponentName2> $=$ *<SubcomponentName2>-new*,
$\ldots |\!)$

wird die externe Ausgabe erzeugt

output-new $=$ *<ComponentName>-make-output subcomps-new*

und der Gesamtkomponentenzustand konstruiert:

ssd-new $= (\!|$
<ComponentName>-SSD-Subcomponents $=$ *subcomps-new*,
<ComponentName>-SSD-OutPorts $=$ *output-new* $|\!)$;
c-new $= (\!|$ *<ComponentName>-Component-SSD* $=$ *ssd-new* $|\!)$

Die gesamte Zustandsübergangsfunktion sieht wie folgt aus:

defs
<ComponentName>-Comp-TransitionFunction-def :
<ComponentName>-Comp-TransitionFunction input c $\equiv$
let
($*$ *Read inputs*:

```
    external input: copy values from interface to subcomponents
    channels:       transport values from subcomponents to subcomponents *)
  <SubcomponentName1>-input =
   <ComponentName>-make-input--<SubcomponentName1> input c;
  <SubcomponentName2>-input =
   <ComponentName>-make-input--<SubcomponentName2> input c;
  ...
  (* Make computation step for every subcomponent *)
  <SubcomponentName1>-new =
   <SubcomponentName1>-Comp-TransitionFunction
    <SubcomponentName1>-input
    (get-<ComponentName>-Subcomponent--<SubcomponentName1> c);
  <SubcomponentName2>-new =
   <SubcomponentName2>-Comp-TransitionFunction
    <SubcomponentName2>-input
    (get-<ComponentName>-Subcomponent--<SubcomponentName2> c);
  ...
  (* Construct new internal state *)
  subcomps-new      = (|
   <ComponentName>-Subcomponent--<SubcomponentName1> = <SubcomponentName1>-new,
   <ComponentName>-Subcomponent--<SubcomponentName2> = <SubcomponentName2>-new,
   ...|);
  (* Copy computed output values to external interface *)
  output-new        = <ComponentName>-make-output subcomps-new;
  (* Construct new SSD state *)
  ssd-new           = (|
   <ComponentName>-SSD-Subcomponents = subcomps-new,
   <ComponentName>-SSD-OutPorts = output-new |);
  (* Construct new component state *)
  c-new             = (| <ComponentName>-Component-SSD = ssd-new |)
 in c-new
```

Die Zustandsübergangsfunktion für eine hierarchisch aufgebaute Komponente C berechnet somit den neuen Komponentenzustand – wie bereits die Zustandsübergangsfunktion für Komponenten, deren Verhalten durch einen Automaten spezifiziert wird – ausgehend von dem Komponentenzustand nach dem letzten Berechnungsschritt und den aktuellen Eingabewerten. Der Berechnungszustandsstrom für einen Eingabestrom *input* und einen Initialzustands *<ComponentName>-Component-InitValue* kann nun mithilfe der generischen Funktion Δ^ω berechnet werden:

```
i-Exec-Comp-Stream
 <ComponentName>-Comp-TransitionFunction  input
 <ComponentName>-Component-InitValue
```

Der Ausgabestrom der Komponente C für eine beliebige Ausgabeextraktionsfunktion *get-Output-Function* (insbesondere auch für ρ_C, mit welcher sich die durch C definierte Stromverarbeitungsfunktion ergibt) kann nun mit Funktionskomposition (bzw. dem *map*-Operator für endliche Ströme) ermittelt werden:

```
get-Output-Function ∘ i-Exec-Comp-Stream
 <ComponentName>-Comp-TransitionFunction  input
 <ComponentName>-Component-InitValue
```

4 Äquivalenz der Isabelle/HOL-Formalisierung der AUTOFOCUS-Semantik

Wir wollen nun die Äquivalenz der theoretischen Definition der AUTOFOCUS-Semantik ([3, 11, 6], [12, Abschnitt 4.1.2] sowie Abschnitt 4.2.1) und der Semantik von AUTOFOCUS-Modellen in der Isabelle/

HOL-Darstellung zeigen. Hierzu verwenden wir die Äquivalenzrelation $\sim$, die für einen Gesamtzustand c einer AUTOFOCUS-Komponente C und einen Gesamtzustand *isa-c* der entsprechenden Isabelle/HOL-Darstellung *C-Comp* angibt, ob sie gleiche Variablen- und Ausgabeportbelegungen haben und die Zustände ihrer Teilkomponenten (bei zusammengesetzter Komponente) bzw. ihrer Zustandsautomaten (bei Komponente mit STD) äquivalent sind. Entitäten elementarer Datentypen (z.B. ganzzahlige Variablen/Ausgabeports) sind genau dann äquivalent, wenn sie gleiche Belegungen haben. Entitäten benutzerdefinierter endlicher Datentypen (z.B. Aufzählungsdatentypen, Kontrollzustände eines Automaten) sind genau dann äquivalent, wenn sie jeweils einander zugeordnete Werte haben (z.B. Zustände $State1$ in AUTOFOCUS und *Automaton1-State1* in der Isabelle/HOL-Darstellung). Entitäten zusammengesetzter Datentypen (z.B. Records, Listen), sind äquivalent, wenn jeder Einzelwert zu dem jeweiligen Einzelwert in der Isabelle/HOL-Darstellung äquivalent ist. Analog ist ein Strom $s \in M^*$ (bzw. $s \in M^\infty$) genau dann zu einem (endlichen bzw. unendlichen) Strom xs in Isabelle/HOL äquivalent, wenn sie gleich lang (bzw. beide unendlich) sind und an jeder Stelle $n \in \mathsf{dom}.s$ äquivalente Elemente besitzen. Schließlich sind eine Transitionsfunktion $\delta_C : I \times C \to C$ und eine Isabelle/HOL-Transitionsfunktion *C-trans-fun* :: *C-SSD-InputPorts-type* $\Rightarrow$ *C-Comp* $\Rightarrow$ *C-Comp* genau dann äquivalent, wenn sie äquivalente Komponentenzustände für äquivalenten Eingaben wieder in äquivalente Zustände überführen.

4.1 Schrittsemantik und Stromverarbeitungssemantik

Die Äquivalenz der Semantikdefinition in [12, Abschnitt 4.1] und der Isabelle/HOL-Darstellung für die Schrittsemantik und die sich daraus ergebenden stromverarbeitenden Funktionen als Komponentensemantik (vgl. Abschnitt 3.1) ergibt sich aus der Isabelle/HOL-Definition der generischen Stromverarbeitungsfunktionen *f-Exec-Comp*, *f-Exec-Comp-Stream*, *i-Exec-Comp-Stream*, welche die entsprechenden Funktionen Δ und Δ^ω darstellen.

Sei C eine AUTOFOCUS-Komponente mit der syntaktischen Schnittstelle $(I \rhd O)$ und der Transitionsfunktion $\delta_C : I \times C \to C$. Seien *C-Comp*, C-SSD-InputPorts-type, C-SSD-OutputPorts-type die Isabelle/HOL-Darstellungen jeweils der Komponente C und ihrer syntaktischen Schnittstelle, sowie *C-trans-fun*:: *C-SSD-InputPorts-type message-af* $\Rightarrow$ *C-Comp* $\Rightarrow$ *C-Comp* die Darstellung ihrer Transitionsfunktion.

Lemma 1 (Äquivalenz der Zustandsstromsemantik) *Sind Komponentenzustände c und isa-c sowie Eingabeströme input und isa-input äquivalent, so sind die Berechnungen in* AUTOFOCUS *und Isabelle/HOL äquivalent:*

$$c \sim \textit{isa-c} \wedge \textit{input} \sim \textit{isa-input} \wedge \delta_C \sim \textit{C-trans-fun} \Rightarrow$$
$$\Delta^\omega(\delta_C,\ \textit{input},\ c) \sim \begin{cases} \textit{f-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \textit{if } \textit{input} \in I^* \\ \textit{i-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \textit{otherwise (i.e., } \textit{input} \in I^\infty\textit{)} \end{cases}$$

BEWEIS Wir unterscheiden die Fälle des endlichen und unendlichen Eingabestroms $input \in I^\omega$.

- Endlicher Eingabestrom: $input \in I^*$.

  Die Verarbeitung einer endlichen Eingabe wird in der Isabelle/HOL-Darstellung mittels der Funktion *f-Exec-Comp-Stream* durchgeführt (vgl. Abschnitt 3.1). Die Behauptung wird induktiv über die Länge des Eingabestroms gezeigt.

  1) Induktionsbeginn: Für einen leeren Eingabestrom $\langle\rangle$ der Länge 0 liefert die Stromverarbeitungsfunktion Δ^ω einen leeren Zustandsstrom (3):

  $$\Delta^\omega(\delta_C,\ \langle\rangle,\ c) = \langle\rangle$$

  Ebenso liefert *f-Exec-Comp-Stream* einen leeren Zustandsstrom:

  f-Exec-Comp-Stream trans-fun [] *isa-c* = []

  Die leeren Ergebnisströme sind trivialerweise äquivalent.

2) Induktionsschritt: Sei die Behauptung für einen endlichen Eingabestrom $input$ der Länge $n \in \mathbb{N}$ und einen (dazu äquivalenten) Eingabestrom *isa-input* in der Isabelle/HOL-Darstellung erfüllt, d.h.,

$$\Delta^{\omega}(\delta_C,\ input,\ c)\ \sim\ \textit{f-Exec-Comp-Stream C-trans-fun isa-input isa-c}$$

Betrachten wir nun den Eingabestrom $input \frown \langle m \rangle$ der Länge $n+1$, der aus dem Eingabestrom $input$ und einer zusätzlichen Eingabe m besteht. Das Berechnungsergebnis besteht aus dem Berechnungsergebnis für $input$ und zusätzlich dem Komponentenzustand nach der Verarbeitung von m:

$$\Delta^{\omega}(\delta_C,\ input \frown \langle m \rangle,\ c)\ =\ \Delta^{\omega}(\delta_C,\ input,\ c) \frown \langle \delta_C(m, \Delta(\delta_C,\ input,\ c)) \rangle$$

Ebenso gilt für die Funktion *f-Exec-Comp-Stream* (vgl. Lemma *f-Exec-Stream-snoc* in Abschnitt 3.1) für einen zu $input \frown \langle m \rangle$ äquivalenten Eingabestrom *isa-input* @ [*isa-m*]:

f-Exec-Comp-Stream C-trans-fun (*isa-input* @ [*isa-m*]) *isa-c* =
f-Exec-Comp-Stream C-trans-fun isa-input isa-c @
[*C-trans-fun isa-m* (*f-Exec-Comp C-trans-fun isa-input c*)]

Die linken Teile $\Delta^{\omega}(\delta_C,\ input,\ c)$ und *f-Exec-Comp-Stream trans-fun-C isa-input isa-c* der konkatenierten Ergebnisströme sind laut der Induktionsannahme äquivalent. Für die rechten Teile der Konkatenationen, in denen die Eingabe m verarbeitet wird, gilt:

- Die Eingaben m und *isa-m* sind äquivalent.
- Die zuletzt erreichten Komponentenzustände $\Delta(\delta_C,\ input,\ c)$ und *f-Exec-Comp C-trans-fun isa-input c* sind äquivalent, da sie jeweils die letzten Elemente der Ergebnisströme $\Delta^{\omega}(\delta_C,\ input,\ c)$ und *f-Exec-Comp-Stream C-trans-fun isa-input c* sind, und diese Ergebnisströme nach der Induktionsannahme äquivalent sind.

Daher liefern die (ihrerseits äquivalenten) Transitionsfunktionen δ_C und *C-trans-fun* für die zusätzliche Eingabe m bzw. *isa-m* wiederum äquivalente Komponentenzustände. Da die linken und die rechten Seiten der konkatenierten Ergebnisströme jeweils äquivalent sind, sind die gesamten Ergebnisströme $\Delta^{\omega}(\delta_C,\ input \frown \langle m \rangle,\ c)$ und f-Exec-Comp-Stream trans-fun-C (input-isa @ [isa-m] äquivalent, womit die Induktionsbehauptung gezeigt ist.

- Unendlicher Eingabestrom: $input \in I^{\infty}$.

  Die Verarbeitung einer unendlichen Eingabe wird in der Isabelle/HOL-Darstellung mittels der Funktion *i-Exec-Comp-Stream* durchgeführt (vgl. Abschnitt 3.1). Die Ergebnisströme sind äquivalent genau dann, wenn für jedes $n \in \mathbb{N}$ ihre jeweiligen Elemente an der Position n äquivalent sind:

$$\forall n \in \mathbb{N}: \quad \Delta^{\omega}(\delta_C,\ input,\ c).n\ \sim\ (\textit{i-Exec-Comp-Stream C-trans-fun isa-input isa-c})\ \textit{n}$$

  Das n-te Element des unendlichen Zustandsstroms ist gleich dem letzten Element des endlichen Zustandsstroms, der bei der Verarbeitung des Eingabestroms bis zum n-ten Element, d.h., des Stroms $\langle input.0, \ldots, input.n \rangle = input{\downarrow_{(n+1)}}$ entsteht (vgl. auch Lemma *i-Exec-Stream-nth* in Abschnitt 3.1). Daher ist zu zeigen:

$$\forall n \in \mathbb{N}: \quad \Delta(\delta_C,\ input{\downarrow_{(n+1)}},\ c)\ \sim\ \textit{f-Exec-Comp C-trans-fun}\ (\textit{isa-input} \Downarrow \textit{Suc n})\ \textit{isa-c}$$

  Die Eingabeströme $input{\downarrow_{(n+1)}}$ und *isa-input* $\Downarrow$ *Suc n* sind endlich und haben die Länge $n+1$ (in Isabelle/HOL wird mit der Funktion *Suc* für eine natürliche Zahl *n* die Nachfolgezahl, d. h., $n+1$ gebildet). Weil die (unendlichen) Ströme $input$ und *isa-input* äquivalent sind, sind insbesondere ihre endlichen Präfixe $input{\downarrow_{(n+1)}}$ und *isa-input* $\Downarrow$ *Suc n* äquivalent. Wie oben für endliche Berechnungen gezeigt, gilt damit:

$$\Delta^{\omega}(\delta_C,\ input{\downarrow_{(n+1)}},\ c)\ \sim\ \textit{f-Exec-Comp-Stream C-trans-fun}\ (\textit{isa-input} \Downarrow \textit{Suc n})\ \textit{isa-c}$$

und damit insbesondere

$$\mathsf{last}(\Delta^{\omega}(\delta_C,\ input{\downarrow_{(n+1)}},\ c))\ \sim\ \textit{last f-Exec-Comp-Stream C-trans-fun}\ (\textit{isa-input} \Downarrow \textit{Suc n})\ \textit{isa-c}$$

Diese Berechnungszustände sind genau die Zustände, die sich nach der Verarbeitung der Eingabeströme bis einschließlich zur Positionen n ergeben (vgl. auch Lemma *f-Exec-eq-f-Exec-Stream-last-if* in Abschnitt 3.1):

$$\Delta(\delta_C,\ input{\downarrow_{(n+1)}},\ c)\ \sim\ \textit{f-Exec-Comp C-trans-fun}\ (\textit{isa-input} \Downarrow \textit{Suc n})\ \textit{isa-c}$$

Damit ist die Äquivalenz der Berechnungszustände für alle $n \in \mathbb{N}$ und foglich für unendliche Eingaben gezeigt.

Wir haben die Äquivalenz der Stromverarbeitung für den Fall der endlichen und den Fall der unendlichen Eingabeströme und somit für alle Eingabeströme $input \in I^*$ gezeigt. $\square$

Daher entspricht die Isabelle/HOL-Darstellung der Stromverarbeitung für endliche und unendliche Ströme der Semantikdefinition durch die Funktionen Δ und Δ^{ω}.

4.2 Verhaltensspezifikation durch Zustandsautomaten

Wir betrachten die Semantik von Komponenten, deren Verhalten durch einen Zustandsautomaten (STD) gegeben wird – diese Komponenten bilden die Blätter des Strukturbaums eines AUTOFOCUS-Modells (vgl. Abschnitt 1). Zum Nachweis der Semantikäquivalenz für AUTOFOCUS und für die Isabelle/HOL-Darstellung von AUTOFOCUS-Modellen wollen wir zunächst die Transitionssemantik von Komponenten mit Verhaltensverfeinerung definieren.

4.2.1 Semantikdefinition

Sei C eine Komponente mit Verhaltensverfeinerung, deren Verhalten durch einen Zustandsautomaten A mit der Zustandsmenge $Q = \{q_1, \ldots, q_{n_Q}\}$ definiert wird. Neben der Kommunikationsschnittstelle $(I \rhd O)$ (vgl. Abschnitt 3.1) kann die Komponente C lokale Variablen $LV = \{v_1, \ldots, v_{n_{LV}}\}$ besitzen. Der lokale Zustand S von C besteht aus den lokalen Variablen LV und dem aktuellen Kontrollzustand $q \in Q$ des Automaten A:

$$S = LV \times Q \tag{6}$$

Der Gesamtzustand von C besteht wiederum aus dem lokalen Zustand S und der Ausgabe O (vgl. Abschnitt 3.1).

Der Automat A definiert für die Komponente C eine Transitionsfunktion $\delta_C : I \times C \to C$, die aus der Eingabe und dem aktuellen Zustand (lokale Variable der Komponente C und Kontrollzustand des Automaten A) den neuen Gesamtzustand der Komponente C berechnet, der den internen Zustand und die Ausgabe enthält. Die Transitionsfunktion δ_C wird durch die Menge τ_A der Zustandsübergänge im Automaten A definiert. Dabei nehmen wir an, dass für jeden Kontrollzustand $q \in Q$ eine totale Ordnung $O_q \subseteq \tau_q \times \tau_q$ auf den aus diesem Zustand ausgehenden Transitionen existiert – diese Ordnung wird zur Reproduzierbarkeit der Übersetzung von Modellen aus AUTOFOCUS in Isabelle/HOL (aber auch andere Zielsprachen) sowie zum Beweis der Semantikäquivalenz der AUTOFOCUS-Modelle und ihrer Isabelle/HOL-Darstellungen benötigt. Eine Transition $t \in \tau_q$, die aus einem Zustand $q \in Q$ ausgeht, ist ein 5-Tupel

$$(InputPattern, PreCondition, PostCondition, OutputPattern, DestinationState) \tag{7}$$

Die ersten beiden Bestandteile bilden die Vorbedingung der Transition

- Das *Eingabemuster* gibt an, welche Ports leer und welche nicht-leer sein müssen.[1] Ferner werden den eingegebenen Portwerten temporäre lokale Bezeichner zugewiesen, die in der Transition verwendet

[1] Das AUTOFOCUS-Werkzeug lässt auch andere Eingabeportmuster zu, wie beispielsweise $p?5$, das erfüllt ist, wenn p den Wert 5 empfängt. Diese Eingabemuster können als abkürzende Schreibweisen für Vorbedingungen betrachtet werden und erhöhen die Ausdrucksstärke der verwendeten Notation nicht (beispielsweise entspricht $p?5$ einem Eingabeportmuster $p?var$ mit der Vorbedingung $var == 5$). Bei der in diesem Bericht behandelten Übersetzung von AUTOFOCUS-Modellen in Isabelle/HOL betrachten wir nur die grundlegenden Eingabemuster $p?var$ und $p?$.

werden können:

$$InputPattern : I \times String \tag{8}$$

Das Eingabemuster hat die Form $p?s$, wobei $p \in I$ ein Eingabeport von C und $s \in String$ eine Zeichenkette ist.

Ein Eingabeportmuster $p?var$ besagt, dass p einen nicht-leeren Wert haben muss ($p \neq \varepsilon$) und dass dieser Wert in anderen Bestandteilen der Transition (d.h., Vorbedingung, Nachbedingung, Ausgabemuster) mit dem lokalen Bezeichner var referenziert werden kann.

Ein Eingabeportmuster der Form $p?$ (mit leerem lokalen Bezeichner) besagt, dass p leer sein muss ($p = \varepsilon$).

Ein Eingabemuster $InputPattern$ passt zu einer Eingabe $m \in I$, wenn jedes Eingabeportmuster $(p, s) \in InputPattern$ dazu passt:

$$\begin{aligned} &InputPattern(m) \;\Leftrightarrow \\ &\quad \forall (p, s) \in InputPattern : \Pi_p(m) = \varepsilon \;\Leftrightarrow\; s = \text{''} \end{aligned} \tag{9}$$

Hierbei passt ein Eingabeportmuster (p, s) genau dann zu einer Eingabe, wenn s und p beide leer sind ($p = \varepsilon$ und $s = \text{''}$), oder s und p beide nicht-leer sind. Taucht ein Eingabeport in keinem der Eingabeportmuster auf, so ist jede (leere oder nicht-leere) Eingabe zulässig.

- Die *Vorbedingung* gibt ein logisches Prädikat an, bei dessen Erfüllung die Transition schalten kann.

$$PreCondition : I \times LV \rightarrow \mathbb{B} \tag{10}$$

Der Einfachheit halber nehmen wir hier und im Weiteren an, dass auf die Werte eines nicht-leeren Eingabeports direkt mit den Portnamen zugegriffen werden kann – für ein Eingabeportmuster $p?var$ wird also anstatt des lokalen Bezeichners var der Portname p verwendet. Eine entsprechende syntaktische Umformung der Vorbedingungen, Nachbedingungen und Ausgabemuster ist trivial, da einfach alle Vorkommen des lokalen Bezeichners var mit dem Portnamen p ersetzt werden. Diese Annahme ist insbesondere mit Hinblick auf die Übersetzung in Isabelle/HOL hilfreich, da in der Isabelle/HOL-Darstellung beim Zugriff auf die Werte nicht-leerer Ports nicht die lokalen Bezeichner sondern direkt die Portnamen verwendet werden.

Eine Transition t kann schalten, wenn ihr Eingabemuster auf die Eingabe passt und die Vorbedingung erfüllt ist. Wir bezeichnen dieses Prädikat als die Gesamtvorbedingung:

$$\begin{aligned} &CompletePreCondition : I \times LV \rightarrow \mathbb{B} \\ &CompletePreCondition(m, lv) \;=\; InputPattern(m) \;\wedge\; PreCondition(m, lv) \end{aligned} \tag{11}$$

Die nächsten zwei Bestandteile einer Transition definieren die Auswirkungen der Transitionsausführung auf die Ausgabe und die lokalen Variablen:

- Das *Ausgabemuster* definiert die Ausgaben am Ende der Transition:

$$OutputPattern : O \times E \tag{12}$$

wobei E die Menge der Ausdrücke über der Eingabe und dem lokalen Zustand der Komponente C ist, deren Ergebnisdatentypen dem Datentyp von mindestens einem Port aus O gleich sind:

$$E \;=\; I \times LV \rightarrow \mathsf{Type}(O) \tag{13}$$

Ein Ausgabeport darf in einem Ausgabemuster höchstens einmal vorkommen, so dass $OutputPattern$ auch als partielle Abbildung von Ausgabeports auf Ausdrücke interpretiert werden kann:

$$OutputPattern : O \rightarrow (E \cup \{\bot\}) \tag{14}$$

Ein Ausgabeportmuster hat die Form $p!expr$, wobei $p \in O$ ein Ausgabeport von C und $expr \in E$ ein Ausdruck ist.[2] Es besagt, dass p einen nicht-leeren Wert ausgibt, der sich bei Auswertung von $expr$ ergibt. Falls p in keinem Ausgabeportmuster vorkommt ($OutputPattern(p) = \bot$), liefert er eine leere Ausgabe:

$$\begin{aligned} &ComputeOutput(OutputPattern, m, lv, p) = \\ &\quad \text{if } OutputPattern(p) = \bot \text{ then } \varepsilon \text{ else let } expr = OutputPattern(p) \text{ in } expr(m, s) \end{aligned} \tag{15}$$

Diese Funktion kann punktweise auf die gesamte Ausgabeschnittstelle $O = (op_1, \ldots, op_{n_{out}})$ erweitert werden:

$$\begin{aligned} &ComputeOutput(OutputPattern, m, lv, O) = (\\ &\quad ComputeOutput(OutputPattern, m, lv, op_1), \\ &\quad \ldots, \\ &\quad ComputeOutput(OutputPattern, m, lv, op_{n_{out}})) \end{aligned} \tag{16}$$

Auf diese Weise können bei Ausführung einer Transition alle in ihrem Ausgabemuster $OutputPattern$ vorkommenden Ausgabeportmuster ausgeführt werden, indem die Ausdrücke ausgewertet und die Ergebnisse an die jeweiligen Ausgabeports geschrieben werden.

- Die *Nachbedingung* definiert die neuen Werte lokaler Variablen am Ende der Transition:

$$PostCondition : LV \times E \tag{17}$$

wobei E, wie bereits bei den Ausgabeports, die Menge der Ausdrücke über der Eingabe und dem lokalen Zustand der Komponente C ist, deren Ergebnisdatentypen dem Datentyp von mindestens einer lokalen Variablen aus LV gleich sind:

$$E \;=\; I \times LV \rightarrow \mathsf{Type}(LV) \tag{18}$$

Analog zu Ausgabeports darf eine lokale Variable in einer Nachbedingung höchstens einmal vorkommen, so dass $PostCondition$ als partielle Abbildung lokaler Variablen auf Ausdrücke interpretiert werden kann:

$$PostCondition : LV \rightarrow (E \cup \{\bot\}) \tag{19}$$

Eine einzelne Nachbedingung aus der Menge $PostCondition$ der Nachbedingungen hat die Form einer Zuweisung $v = expr$, wobei $v \in VL$ eine lokale Variable von C und $expr \in E$ ein Ausdruck ist. Sie besagt, dass v einen neue Wert erhält, der sich bei Auswertung von $expr$ ergibt. Falls v in keiner Zuweisung vorkommt ($PostCondition(v) = \bot$), wird ihr Wert nicht verändert:

$$\begin{aligned} &ComputeLocalVariable(PostCondition, m, lv, v) = \\ &\quad \text{if } PostCondition(v) = \bot \text{ then } \Pi_v(lv) \text{ else} \\ &\quad \text{let } expr = PostCondition(v) \text{ in } expr(m, lv) \end{aligned} \tag{20}$$

Diese Funktion kann punktweise auf alle lokalen Variablen $LV = (v_1, \ldots, v_{n_{LV}})$ erweitert werden:

$$\begin{aligned} &ComputeLocalVariable(PostCondition, m, s, LV) = (\\ &\quad ComputeLocalVariable(PostCondition, m, s, v_1), \\ &\quad \ldots, \\ &\quad ComputeLocalVariable(PostCondition, m, s, v_{n_{LV}})) \end{aligned} \tag{21}$$

[2] Das Ausgabemuster $p!$ wird vom AUTOFOCUS-Werkzeug zwar zugelassen, hat jedoch keine Auswirkung auf die Ausgabe, da jedes Ausgabeport nur einmal beschrieben werden kann (Ausgaben können also nicht innerhalb eines Ausgabemusters überschrieben werden, insbesondere auch nicht mit einer leeren Ausgabe durch $p!$), und ein Ausgabeport, der nicht im Ausgabemuster enthalten ist, automatisch eine leere Nachricht ausgibt. Daher ist das Ausgabemuster $p!$ für die Semantikdefinition und die Übersetzung in Isabelle/HOL nicht relevant.

Auf diese Weise können bei Ausführung einer Transition alle in ihrer Nachbedingung $PostCondition$ vorkommenden Zuweisungen ausgeführt werden, indem die Ausdrücke ausgewertet und die Ergebnisse den jeweiligen lokalen Variablen zugewiesen werden.

Der Kontrollzustand, in den die Komponente nach der Ausführung der Transition wechselt, wird durch den *Zielzustand* im 5-Tupel einer Transition angegeben.

Somit wird durch die Ausführung einer Transition t die Komponente C in den neuen Zustand überführt, der mit folgender Funktion berechnet wird:

$$\begin{aligned}
&ComputeComponentState\ : I \times LV \rightarrow C \\
&ComputeComponentState(m, lv)\ =\ \kappa_C\big(\\
&\quad (ComputeLocalVariable(PostCondition, m, lv, LV),\ DestinationState), \\
&\quad ComputeOutput(OutputPattern, m, lv, O)\big)
\end{aligned} \tag{22}$$

Nachdem die Vorbedingung und die Auswirkung der Ausführung einer Transition formalisiert sind, können wir die durch A gegebene Transitionsfunktion δ_C der Komponente C definieren. Zunächst muss diejenige Transition ausgewählt werden, die vom aktuellen Kontrollzustand q ausgeht, deren vollständige Vorbedingung erfüllt ist, und die bezüglich der Ordnung O_q minimal ist. Falls für keine Transition die vollständige Vorbedingung erfüllt ist, wird $\perp$ zurückgegeben.

$$\begin{aligned}
&SelectTransition : Q \times I \times LV \rightarrow (\tau_A\ \cup \{\perp\}) \\
&SelectTransition(q, m, lv) = \\
&\quad \mathsf{min}_{(O_q)}\{\, t \mid t \in \tau_q\ \wedge\ CompletePreCondition(m, lv)\,\}
\end{aligned} \tag{23}$$

Nun kann die Transitionsfunktion von C definiert werden.

$$\begin{aligned}
&\delta_C(m, c) = \\
&\mathsf{let} \\
&\quad q = \Pi_Q(\Pi_S(c));\ \text{/* Current control state */} \\
&\quad lv = \Pi_{LV}(\Pi_S(c));\ \text{/* Current values of local variables */} \\
&\quad t = SelectTransition(q, m, lv) \\
&\mathsf{in} \\
&\quad \mathsf{if}\ t = \perp\ \mathsf{then}\ FlushOutput(c)\ \mathsf{else}\ ComputeComponentState_t(m, lv)
\end{aligned} \tag{24}$$

wobei $FlushOutput$ die Ausgabeports einer Komponente leert und ihren Zustand ansonsten nicht verändert:

$$\begin{aligned}
&FlushOutput : C \rightarrow C \\
&FlushOutput(c)\ =\ \kappa_C(\sigma_C(c), \varepsilon)
\end{aligned} \tag{25}$$

4.2.2 Semantikäquivalenz

Wir wollen nun die Äquivalenz der durch den Zustandsautomaten A gegebenen Transitionsfunktion δ_C für die AUTOFOCUS-Komponente C, und der Transitionsfunktion *C-trans-fun* für ihre Isabelle/HOL-Darstellung (Abschnitt 3.2) beweisen.

Wir nehmen an, dass für jeden Zustand $q_i \in Q_A$ die aus ihm ausgehenden Transitionen entsprechend der Ordnungsrelation O_{q_i} nummeriert sind:

$$\begin{aligned}
&\tau_{q_i} = \{t_{i,1},\ \ldots,\ t_{i,n_i}\} \\
&\forall k \in \{1, \ldots, n-1\} : O_{q_i}(t_{i,k},\ t_{i,k+1})
\end{aligned} \tag{26}$$

Zudem bezeichnen wir für eine Transition $t_{i,k} \in \tau_{q_i}$ mit

$$R^{Isa}_{t_{i,k}} \text{ für } R_{t_{i,k}} \in \{PreCondition_{t_{i,k}},\ PostCondition_{t_{i,k}},\ OutputPattern_{t_{i,k}}\}$$

die Vorbedingungen, Ausgabemuster und Nachbedingungen, in denen Prädikate und Berechnungsergebnisse anstatt der in AUTOFOCUS verwendeten QuestF-Ausdrücke durch Isabelle/HOL-Ausdrücke gegeben werden, die auch in der Isabelle/HOL-Darstellung *A-STD-Transition-Q-i--Q-j-k--k* benutzt werden. Für das Ausgabemuster und Nachbedingung wird die Äquivalenz dieser Ausdrücke, wie bereits für Ströme und komplexe Datentypen (vgl. Beginn des Abschnitts 4), punktweise definiert:

$$\begin{aligned} &ExpressionsEquiv(t_{i,k},\ \textit{A-STD-Transition-Q-i--Q-j-k--k}) \ \Leftrightarrow \\ &\quad PreCondition_{t_{i,k}} \sim PreCondition^{Isa}_{t_{i,k}} \wedge \\ &\quad \forall v \in LV : PostCondition_{t_{i,k}}(v) \sim PostCondition^{Isa}_{t_{i,k}}(v) \wedge \\ &\quad \forall p \in O : OutputPattern_{t_{i,k}}(p) \sim OutputPattern^{Isa}_{t_{i,k}}(p) \end{aligned} \tag{27}$$

Lemma 2 (Äquivalenz der Transitionssemantik von Komponenten mit einem Zustandsautomaten) *Seien Komponentenzustände c und isa-c sowie Eingaben m und isa-m äquivalent. Ferner seien für alle Transitionen $t_{i,k} \in \tau_{q_i}$ für Zustände $q_i \in Q_A$ die in QuestF formulierten Ausdrücke (Vorbedingung, Ausdrücke in Ausgabeportmustern sowie in Zuweisungen in Nachbedingungen) äquivalent zu den jeweiligen Bestandteilen in den entsprechenden Isabelle/HOL-Transitionsfunktionen A-STD-Transition-Q-i--Q-j-k--k. Dann sind die Ergebnisse eines Berechnungsschritts von C in* AUTOFOCUS *und Isabelle/HOL äquivalent:*

$$\begin{aligned} &c \sim \textit{isa-c} \wedge m \sim \textit{isa-m} \wedge \\ &\forall q_i \in Q_A : \forall t_{i,k} \in \tau_{q_i} : ExpressionsEquiv(t_{i,k},\ \textit{A-STD-Transition-Q-i--Q-j-k--k}) \ \Rightarrow \\ &\quad \delta_C(m, c) \sim \textit{C-trans-fun isa-m isa-c} \end{aligned}$$

BEWEIS Die Berechnung der Zusammengesetzten Komponente C und ihrer Isabelle/HOL-Darstellung besteht aus zwei Teilschritten:

1) Auswahl der auszuführenden Transition des Automaten A.

2) Berechnung des neuen Komponentenzustands durch Ausführung der ausgewählten Transition.

Nun zeigen wir für jeden Teilschritt, dass die AUTOFOCUS-Komponente und ihre Isabelle/HOL-Darstellung in äquivalente Komponentenzustände ohne Berücksichtigung der Ausgabe und nach dem letzten Teilschritt auch mit Berücksichtigung der Ausgabe überführt werden.

Bei Komponenten mit Verhaltensverfeinerung (d.h., mit einem Automaten) wird die Eingabe direkt übernommen. Nach Voraussetzung ist $m \sim$ *isa-m*, so dass die Eingabe für C und *C-Comp* äquivalent ist. Das Leeren der Ausgabeports in der Isabelle/HOL-Darstellung *C-Comp* spielt für die Berechnung keine Rolle, da die Belegungen der Ausgabeports nicht bei der Berechnung verwendet werden (die Transitionsfunktionen *A-STD-Transition-Q-i--Q-j-k--k* können zwar aufgrund ihrer Signatur auf die Belegungen der Ausgabeports zugreifen, dies geschieht jedoch aufgrund des Übersetzungsschemas nicht (vgl. Abschnitt 3.2.2), gemäß dem die Transitionsfunktion aus den AUTOFOCUS-Transitionen erzeugt wird, die ihrerseits nicht auf Belegungen der Ausgabeports lesend zugreifen können). Daher wird das Leeren der Ausgabeports erst am Ende des Berechnungsschritts bei der Berechnung der Ausgabe betrachtet.

Die Transitionsfunktion δ_C führt folgende Schritte durch:

1) Auswahl der Transition

   Die Transitionsfunktion δ_C (24) wählt die auszuführende Transition mithilfe der Funktion $Select\text{-}Transition$ (23) aus. Diese wählt diejenige Transition, deren Gesamtvorbedingung (11) erfüllt ist, und die bezüglich der Ordnung O_{q_i} für den aktuellen Zustand $q_i \in Q_A$ minimal ist.

   Die Gesamtvorbedingung $CompletePreCondition$ besteht aus dem Eingabemuster $InputPattern$ und der eigentlichen Vorbedingung $PreCondition$.

   - Das Eingabemuster $InputPattern$ (9) unterscheidet drei Fälle für einen Eingabeport $p \in I$:

- p kommt in einem Eingabemuster $p?x$ (mit einem nicht-leeren Bezeichner x) vor. In diesem Fall wird für die Isabelle/HOL-Darstellung *A-STD-TransitionFunction-selectTransition* der Transitionsauswahlfunktion eine Vorbedingung $p \neq \varepsilon$ erzeugt. Äquivalent dazu fordert (9), dass p nicht-leer ist, weil der Bezeichner x nicht-leer ist.
- p kommt in einem Eingabemuster $p?$ vor, d.h. $(p, '') \in InputPattern$. In diesem Fall wird für *A-STD-TransitionFunction-selectTransition* eine Vorbedingung $p = \varepsilon$ erzeugt. Äquivalent dazu fordert (9), dass p leer ist, weil dem Port p im Eingabemuster ein leerer Bezeichner $''$ zugeordnet ist.
- p kommt in keinem Eingabemuster vor. Dann wird in *A-STD-TransitionFunction-selectTransition* für p keine Vorbedingung generiert. Äquivalent dazu stellt (9) keine Anforderungen an Eingabeports, die in keinem Eingabemuster vorkommen.

Damit sind die aus dem Eingabemuster einer Transition für *A-STD-TransitionFunction-selectTransition* generierten Vorbedingungen äquivalent zu seiner durch (9) definierten Semantik.

- Nach Voraussetzung des Lemmas sind die in QuestF formulierten Ausdrücke äquivalent zu den jeweiligen Bestandteilen in den entsprechenden Isabelle/HOL-Transitionsfunktionen. Insbesondere ist auch die Vorbedingung $PreCondition$ äquivalent zu der für *A-STD-TransitionFunction-selectTransition* erstellten Vorbedingung.

Die Gesamtvorbedingung entspricht sowohl in AUTOFOCUS als auch in der Isabelle/HOL-Darstellung der Konjunktion der Eingabemuster und der Transitionsvorbedingung (vgl. (11) und Definition von *A-STD-TransitionFunction-selectTransition* im Abschnitt 3.2.2). Da beide Bestandteile jeweils äquivalent sind, ist auch die Gesamtvorbedingung $CompletePreCondition$ einer Transition äquivalent zu der für diese Transition generierten Gesamtvorbedingung in der Transitionsauswahlfunktion *A-STD-TransitionFunction-selectTransition*.

Gemäß (23) wählt $SelectTransition$ die bezüglich O_{q_i} minimale Transition $t_{i,k}$, deren Gesamtvorbedingung erfüllt ist. In *A-STD-TransitionFunction-selectTransition* sind die *if*-Anweisungen entsprechend der Ordnung O_{q_i} angeordnet. Damit wird die $t_{i,k}$ entsprechende Transitionsfunktion *A-STD-Transition-Q-i--Q-j-k--k* ausgewählt, weil die entsprechende *if*-Anweisungen die erste ist, für welche die Gesamtvorbedingung erfüllt ist.

Falls für keine Transition die Gesamtvorbedingung erfüllt ist und $SelectTransition$ $\perp$ zurückgibt, dann wählt *A-STD-TransitionFunction-selectTransition* die Transitionsfunktion *A-STD-NoTransition*, die keine Änderungen an dem Komponentenzustand vornimmt – dieser Fall wird weiter unten bei der Behandlung der Transitionsausführung betrachtet.

2) Ausführung der ausgewählten Transition

Wir betrachten zunächst den Fall, dass die Transitionsauswahlfunktion eine Transition $t_{i,k}$ ausgewählt hat ($SelectTransition = t_{i,k}$). Dann wird diese Transition ausgeführt – dabei werden die Ausgabewerte und die neuen Werte lokaler Variablen berechnet sowie der neue Kontrollzustand gesetzt. Der neue Komponentezustand wird durch die Funktion $ComputeComponentState$ (22) ermittelt. Dabei werden Werte für folgende Elemente des Komponentenzustands berechnet:

- Die Ausgabe wird durch die Funktion $ComputeOutput$ auf Grundlage des Ausgabemusters $OutputPattern$ aus der Eingabe und den aktuellen (noch nicht neu berechneten) Werten lokaler Variablen berechnet (15), (16). Es sind zwei Fälle zu unterscheiden:
  - p kommt in einem Ausgabeportmuster $p!expr$ vor. In diesem Fall wird für die Isabelle/HOL-Darstellung *A-STD-Transition-Q-i--Q-j-k--k* der Transitionsfunktion die Zeile

    set-C-OutPort-p (*Msg <Computed result for p>*) ...

    erzeugt, wobei *<Computed result for p>* die Isabelle/HOL-Darstellung des QuestF-Ausdrucks $expr$ bezeichnet, die nach Voraussetzung äquivalent zu $expr$ ist. Die Funktion $ComputeOutput$ berechnet für p den Ausgabewert durch Auswertung des Ausdrucks $expr$ (15), womit die Ergebnisse äquivalent sind.

– p kommt in keinem Ausgabeportmuster vor. Dann wird in *A-STD-Transition-Q-i--Q-j-k--k* keine Zeile für p generiert, so dass p den leeren Ausgabewert ε behält, der zu Beginn der Transition durch die Funktion *flushOutputPorts-C-SSD* an alle Ausgabeports geschrieben wurde (vgl. Definition der Komponententransitionsfunktion *C-Comp-TransitionFunction* im Abschnitt 3.2.3 sowie Definition von *flushOutputPorts-C-SSD* im Abschnitt 3.2.1). Äquivalent dazu schreibt die Funktion $ComputeOutput$ eine leere Nachricht an den Ausgabeport p (15), da p in keinem Ausgabeportmuster vorkommt und deshalb $Output\text{-}Pattern(p) = \bot$ ist.

Damit sind die entsprechend dem Ausgabemuster einer Transition $t_{i,k}$ gemäß (15), (16) berechneten Ausgabewerte äquivalent zu den Ausgabewerten in der Isabelle/HOL-Darstellung nach der Ausführung der Transition *A-STD-Transition-Q-i--Q-j-k--k*.

- Die neuen Werte der lokalen Variablen werden durch die Funktion $ComputeLocalVariable$ ebenfalls aus der Eingabe und den aktuellen Werten lokaler Variablen berechnet (20), (21). Es sind wiederum zwei Fälle zu unterscheiden:
  – v kommt in der Nachbedingung in einer Zuweisung $v = expr$ auf der linken Seite vor. In diesem Fall wird für die Isabelle/HOL-Darstellung *A-STD-Transition-Q-i--Q-j-k--k* der Transitionsfunktion die Zeile

    set-C-LocVar-v <Computed result for v> ...

    erzeugt, wobei *<Computed result for v>* die Isabelle/HOL-Darstellung des QuestF-Ausdrucks $expr$ bezeichnet, die nach Voraussetzung äquivalent zu $expr$ ist. Die Funktion $ComputeLocalVariable$ berechnet für v den neuen Wert durch Auswertung des Ausdrucks $expr$ (20), womit die Ergebnisse äquivalent sind.
  – v kommt in der Nachbedingung in keiner Zuweisung auf der linken Seite vor. Dann wird in *A-STD-Transition-Q-i--Q-j-k--k* keine Zeile für v generiert, so dass v den vorherigen Wert behält. Äquivalent dazu gibt die Funktion $ComputeLocalVariable$ den bereits vorhandenen Variablenwert für den neuen Komponentenzustand zurück (20), da v in keiner Nachbedingung der Form $(v,\ expr)$ vorkommt und deshalb $PostCondition(v) = \bot$ ist.

  Damit sind die entsprechend der Nachbedingung einer Transition $t_{i,k}$ gemäß (20), (21) berechneten Werte lokaler Variablen äquivalent zu den Variablenwerten in der Isabelle/HOL-Darstellung nach der Ausführung der Transition *A-STD-Transition-Q-i--Q-j-k--k*.

- Schließlich wird der Wert des Kontrollzustands auf den Zielzustand gesetzt. Der Zielzustand ist für eine Transition $t_{i,k}$ fest und wird durch $DestinationState = q_{j_k}$ mit $q_{j_k} \in Q_A$ gegeben. In der Transitionsfunktion *A-STD-Transition-Q-i--Q-j-k--k* wird der Zielzustand ebenfalls mit *A-Q-j-k* angegeben, so dass die Isabelle/HOL-Darstellung der Komponente C bei Ausführung der Transition in den Kontrollzustand überführt wird, der q_{j_k} entspricht.

Die Ausgabe, die Belegungen lokaler Variablen und der Kontrollzustand nach der Ausführung der Transition $t_{i,k}$ sind jeweils äquivalent zu den Werten in der Isabelle/HOL-Darstellung nach der Ausführung der Transitionsfunktion *A-STD-Transition-Q-i--Q-j-k--k*.

Betrachten wir nun den Fall, dass keine Transition ausgewählt wurde, weil für keine Transition, die aus dem Zustand q_i ausgeht, die Gesamtvorbedingung erfüllt ist ($SelectTransition = \bot$). Dann wurde durch den Berechnungsschritt gemäß (24) lediglich die Ausgabe der Komponente geleert: $\delta_C(m, c) = FlushOutput(c)$. Die Auswahlfunktion *A-STD-TransitionFunction-selectTransition* wählte in diesem Fall die leere Transition *A-STD-NoTransition*, die keine Änderungen an dem Komponentenzustand vornimmt – dadurch entspricht der Komponentenzustand nach der Ausführung der Transitionsfunktion *A-STD-TransitionFunction* dem Komponentenzustand vor der Transition, bei dem lediglich die Ausgabe geleert wurde, weil zu Beginn eines Berechnungsschritts die Funktion *flushOutputPorts-C-SSD* ausgeführt wird (vgl. Definition der Funktionen *C-Comp-Transition-Function* im Abschnitt 3.2.3, *A-STD-NoTransition* im Abschnitt 3.2.2 sowie *flushOutputPorts-C-SSD* im Abschnitt 3.2.1) Damit sind die Ergebnisse der Transitionsausführung in AUTOFOCUS und Isabelle/HOL auch in dem Fall äquivalent, dass für keine Transition die Gesamtvorbedingung erfüllt ist.

Für die Komponententransitionsfunktion δ_C und ihre Isabelle/HOL-Darstellung *C-Comp-TransitionFunction* gilt:

1) Die Transitionsauswahlfunktionen $Select\,Transition$ und *A-STD-TransitionFunction-selectTransition* wählen im ersten Teilschritt bei äquivalenten internen Komponentenzuständen und äquivalenten Eingaben einander entsprechende Transitionsfunktionen $t_{i,k}$ und *A-STD-Transition-Q-i--Q-j-k--k* bzw. $\perp$ und *A-STD-NoTransition*, falls für keine Transition die Vorbedingungen erfüllt sind.

2) Die Berechnungsergebnisse nach der Ausführung der ausgewählten Transitionen im zweiten Teilschritt sind äquivalent:

   - Ist die Gesamtvorbedingung für eine Transition $t_{i,k}$ aus dem Zustand q_i erfüllt und ist $t_{i,k}$ unter allen Transitionen, deren Gesamtvorbedingung erfüllt ist, bezüglich der Ordnung O_{q_i} minimal, so wurden die Transitionsfunktionen $t_{i,k}$ und *A-STD-Transition-Q-i--Q-j-k--k* ausgewählt. Diese sind äquivalent unter der Bedingung, dass vor der Ausführung von *A-STD-Transition-Q-i--Q-j-k--k* die Ausgabeports durch *flushOutputPorts-C-SSD* geleert wurden (diese Bedingung wird durch die Konstruktion der Transitionsfunktion *C-Comp-TransitionFunction* erfüllt, vgl. Abschnitt 3.2.3).
   - Sind die Vorbedingungen für keine Transition erfüllt, so werden sowohl in AUTOFOCUS als auch in der Isabelle/HOL-Darstellung nur die Ausgabeports geleert, so dass die Ergebniszustände wiederum äquivalent sind.

Folglich sind die Komponentenzustände nach der Ausführung der gesamten Komponententransitionsfunktionen δ_C und *C-Comp-TransitionFunction* äquivalent. □

4.3 Kompositions- und Kommunikationssemantik

Wir befassen uns nun mit der Semantik einer hierarchisch aufgebauten Komponente, wie sie in [12, Abschnitt 4.1.2] definiert wurde. Der lokale Zustand S einer hierarchisch aufgebauten Komponente C besteht aus den Zuständen ihrer Teilkomponenten $C_1, \ldots, C_n$ (einschließlich der Ausgabe aus dem letzten Berechnungsschritt), die zur Berechnung des nächsten Zustands notwendig sind:

$$S = C_1 \times \cdots \times C_n \tag{28}$$

Der Gesamtzustand von C setzt sich damit aus den Zuständen der Teilkomponenten und der Ausgabe zusammen:

$$C = (S, O) = (C_1 \times \cdots \times C_n, O) \tag{29}$$

Die Semantik einer hierarchisch aufgebauten Komponente C ergibt sich aus Zustandsübergangsfunktionen ihrer Teilkomponenten $\delta_{C_1}, \ldots, \delta_{C_n}$ und der Kommunikationsstruktur $\mathcal{CH}_C$, $\mathcal{IA}_C$, $\mathcal{OA}_C$ [12, Abschnitt 4.1.2] – die externen Eingaben an C sowie die früheren Ausgaben der Teilkomponenten werden entsprechend der Kommunikationsstruktur an die Teilkomponenten verteilt, diese führen jeweils einen Berechnungsschritt aus, und die Ausgaben werden an die externe Schnittstelle weitergegeben.

Wir wollen nun die Äquivalenz der Transitionsfunktion δ_C für die AUTOFOCUS-Komponente C und der Transitionsfunktion *C-trans-fun* für ihre Isabelle/HOL-Darstellung (Abschnitt 3.3) beweisen.

Lemma 3 (Äquivalenz der Transitionssemantik zusammengesetzer Komponenten) *Sind Komponentenzustände* c *und isa-c, Eingaben* m *und isa-m sowie die Zustandsübergangsfunktionen* δ_{C_i} *und C-i-trans-fun der Teilkomponenten* C_i *(*$i \in \{1, \ldots, n\}$*) äquivalent, so sind die Ergebnisse eines Berechnungsschritts von* C *in* AUTOFOCUS *und Isabelle/HOL äquivalent:*

$$c \sim \textit{isa-c} \wedge m \sim \textit{isa-m} \wedge \forall i \in \{1, \ldots, n\} : \delta_{C_i} \sim \textit{C-i-trans-fun} \Rightarrow$$
$$\delta_C(m, c) \sim \textit{C-trans-fun isa-m isa-c}$$

BEWEIS Die Berechnung der zusammengesetzten Komponente C und ihrer Isabelle/HOL-Darstellung besteht aus folgenden Teilschritten:

1) Einlesen der Eingaben:

   - Übertragung der externen Eingaben der Komponente C an die Teilkomponenten C_i ($i \in \{1, \ldots, n\}$) gemäß den Zuordnungen aus $\mathcal{IA}_C$.
   - Übertragung der Ausgaben der Teilkomponenten C_i aus dem letzten Schritt über Kommunikationskanäle aus $\mathcal{CH}_C$ an Eingabeports der Teilkomponenten.

2) Berechnungsschritt für jede Teilkomponente C_i mittels der entsprechenden Transitionsfunktion δ_{C_i}.

3) Ausgabe der Berechnungsergebnisse der Teilkomponenten C_i an die externe Schnittstelle der Komponente C gemäß den Zuordnungen aus $\mathcal{OA}_C$.

Nun zeigen wir für jeden Teilschritt, dass die AUTOFOCUS-Komponente und ihre Isabelle/HOL-Darstellung in äquivalente Komponentenzustände überführt werden.

1) Eingabe

   Das Einlesen der Eingaben für die Teilkomponenten C_i von C ($i \in \{1, \ldots, n\}$) erfolgt in AUTOFOCUS mittels der Funktionen $\mathcal{I}n\mathcal{V}al_{I_i}$ (wobei I_i die Eingabeschnittstelle von C_i ist) und in Isabelle/HOL mittels der Funktionen *C-make-input--C-i* (Abschnitt 3.3.1).

   Die Funktion $\mathcal{I}n\mathcal{V}al_{I_i}$ ermittelt die Eingabe für die Komponente C_i. Sie stellt die punktweise Erweiterung der Funktion $\mathcal{I}n\mathcal{V}al_p$ dar, die für einen Eingabeport p der Teilkomponente den Eingabewert aus der externen Eingabe i_C der Elternkomponente C und dem vorherigen internen Zustand s_C der Komponente C ermittelt (s_C enthält die vorherigen Berechnungsergebnissen der Teilkomponenten C_i). Sie unterscheidet für einen Eingabeport p die Fälle, dass p entweder Zielport eines Kanals, oder mit einem Eingabeport der Elternkomponente assoziiert, oder frei ist – im ersten Fall liefert sie den Ausgabewert aus dem vorherigen Schritt am verbundenen Ausgabeport, im zweiten den Eingabewert am assoziierten Eingabeport von C, und im letzten Fall die leere Nachricht:

$$\mathcal{I}n\mathcal{V}al_p(i_C, s_C) \stackrel{\text{def}}{=} \begin{cases} \Pi_{p_{out}}(\rho_{C_j}(\Pi_j(s_C))) & \text{falls } (p_{out}, p) \in \mathcal{CH}_C \ \wedge \ p_{out} \in \mathcal{OP}_{C_j} \\ \Pi_{p_{in}}(i_C) & \text{falls } (p_{in}, p) \in \mathcal{IA}_C \\ \varepsilon & \text{sonst} \end{cases} \tag{30}$$

   Betrachten wir nun diese drei Fälle bei der Funktion $\mathcal{I}n\mathcal{V}al_{I_i}$ und der Isabelle/HOL-Funktion *C-make-input--C-i*, für einen Eingabeport p_k von C_i.

   1) Der Port p_k ist mit einem Kanal ch verbunden: $p = \mathsf{DestinationPort}(ch)$. Sei $p_l = \mathsf{SourcePort}(ch)$ der Ausgabeport einer Teilkomponente C_j, mit dem p_k durch ch verbunden ist. Dann gilt:

      $$\mathcal{I}n\mathcal{V}al_{p_k}(i_C, s_C) \ = \ \Pi_{p_l}(\rho_{C_j}(\Pi_j(s_C)))$$

      Mit $\Pi_j(s_C))$ wird der Zustand von C_j aus s_C extrahiert, mit $\rho_{C_j}(\Pi_j(s_C))$ die Ausgabe von C_j und mit $\Pi_{p_l}(\rho_{C_j}(\Pi_j(s_C)))$ die Ausgabe von p_l und somit die Eingabe von p_k ermittelt.
      In Isabelle/HOL wird der Eingabewert für p_k durch *C-make-input--C-i* wie folgt ermittelt:

      ```
      C-i-InPort-P-k =
       get-C-j-OutPort-P-l (
        get-C-Subcomponent--C-j isa-c)
      ```

      Mit *get-C-Subcomponent--C-j c* wird der Zustand von C_j aus dem Gesamtzustand von C extrahiert und mit *get-C-j-OutPort-P-l* die Ausgabe von p_l ermittelt. Da die Gesamtzustände c und *isa-c* äquivalent sind, sind auch die Werte an dem Ausgabeports p_l von C_j sowie in Isabelle/HOL an dem Port *C-j-InPort-P-l* äquivalent, und damit sind nach dem Einlesen der Eingabe auch die Werte an dem Eingabeport p_k von C_i sowie dem Port *C-i-InPort-P-k* äquivalent.

   2) Der Port p_k ist mit einem Eingabeport p_l der Oberkomponente C von C_i assoziiert: $(p_k, p_l) \in \mathcal{IA}_C$. Der an p_k anliegende Wert ist in diesem Fall stets gleich dem Wert an p_l:

      $$\mathcal{I}n\mathcal{V}al_{p_k}(i_C, s_C) \ = \ \Pi_{p_l}(i_C)$$

      In Isabelle/HOL wird der Eingabewert für p_k durch *C-make-input--C-i*, wie folgt ermittelt:

C-i-InPort-P-k =
C-InPort-P-1 (the-C-SSD-InputPorts isa-m)

Da die Eingaben m und *isa-m* äquivalent sind, erhalten der Port p_k und seine Isabelle/HOL-Darstellung *C-i-InPort-P-k* äquivalente Werte.

3) Der Port p_k ist frei, d.h., er ist weder mit einem anderen Port durch einen Kanal verbunden, noch mit einem Eingabeport der Schnittstelle von C assoziiert. An einem solchen Eingabeport kommen keine Eingaben an, so dass er stets leer ist:

$$\mathcal{I}n\mathcal{V}al_{p_k}(i_C, s_C) \stackrel{\text{def}}{=} \varepsilon$$

In der Isabelle/HOL-Darstellung wird dem Eingabeport ebenfalls eine leere Nachricht zugewiesen:

C-i-InPort-P-k $= \varepsilon$

Sowohl p_k als auch *C-i-InPort-P-k* enthalten leere Nachricht, so dass ihre Inhalte äquivalent sind.

Somit ermittelt der erste Teilschritt eines Zustandsübergangs – das Einlesen der Eingabe – für die AUTOFOCUS-Komponente C und ihre Isabelle/HOL-Darstellung *C-Comp* äquivalente Eingaben, d.h., für jeden Eingabeport p_k einer Teilkomponente C_i und die entsprechende Isabelle/HOL-Darstellung *C-i-InPort-P-k*, die im nachfolgenden Teilschritt zur Berechnung des neuen Zustands der Teilkomponente C_i verwendet wird, gilt:

$$p_k \sim \textit{C-i-InPort-P-k} \tag{31}$$

Damit gilt für die Eingabe m_i einer Teilkomponente C_i bzw. in Isabelle/HOL für die Eingabe *C-i-input*:

$$I_i \sim \textit{C-i-input} \tag{32}$$

2) Berechnung

Die Transitionsfunktion der Komponente C führt im zweiten Teilschritt die Berechnungsschritte der Teilkomponenten $C_1, \ldots, C_n$, indem sie mittels der Hilfsfunktion $\delta_{S_C} : I \times C \to S_C$ den neuen lokalen Zustand von C berechnet:

$$\begin{aligned} \delta_{S_C}(m, c) \;=\; & \big(\delta_{C_1}(\mathcal{I}n\mathcal{V}al_{I_1}(m, \sigma_C(c)), \Pi_1(\sigma_C(c))), \\ & \ldots, \\ & \delta_{C_n}(\mathcal{I}n\mathcal{V}al_{I_n}(m, \sigma_C(c)), \Pi_n(\sigma_C(c)))\big) \end{aligned} \tag{33}$$

In der Isabelle/HOL-Darstellung der Transitionsfunktion entspricht dies der Konstruktion des neuen lokalen Zustands (Abschnitt 3.3.2):

C-1-new = *C-1-trans-fun C-1-input* (*get-C-Subcomponent--C-1 c*);
C-2-new = *C-2-trans-fun C-2-input* (*get-C-Subcomponent--C-2 c*);
...
subcomps-new = ⦇
C-Subcomponent--C-1 = *C-1-new*,
C-Subcomponent--C-2 = *C-2-new*,
...⦈;

Für jede Teilkomponente C_i wird damit folgende Berechnung durchgeführt:

$$\delta_{C_i}(\mathcal{I}n\mathcal{V}al_{I_i}(m, \sigma_C(c)), \Pi_i(\sigma_C(c)))$$

In der Isabelle/HOL-Darstellung wird folgender neuer Zustand für die Teilkomponente C_i berechnet:

C-i-new = *C-i-trans-fun C-i-input* (*get-C-Subcomponent--C-i c*)

Nach Voraussetzung sind die Komponentenzustände c und *isa-c* äquivalent, so dass insbesondere die darin enthaltenen Zustände der Teilkomponenten C_i und *isa-c-i* äquivalent sind. Ebenso sind die Transitionsfunktionen δ_{C_i} und *C-i-trans-fun* äquivalent. Schließlich sind, wie oben gezeigt, die Eingaben $I_i = \mathcal{I}n\mathcal{V}al_{I_i}(m, \sigma_C(c))$ und *C-i-input* äquivalent (32). Folglich sind auch die neuen Zustände $\delta_{C_i}(\mathcal{I}n\mathcal{V}al_{I_i}(m, \sigma_C(c)), \Pi_i(\sigma_C(c)))$ und *C-i-new* der Teilkomponente C_i äquivalent. Da die obige Überlegung für alle $i \in \{1, \ldots, n\}$ gilt, sind die aus den neuen Teilkomponentenzuständen konstruierten lokalen Zustände $\delta_{S_C}(m, c)$ und *subcomps-new* äquivalent:

$$\delta_{S_C}(m, c) \sim \textit{subcomps-new} \tag{34}$$

3) Ausgabe

Die Ausgabe der Berechnungsergebnisse an der externen Schnittstelle O von C erfolgt in AUTOFOCUS mittels der Funktion $\mathcal{O}ut\mathcal{V}al_O$ und in Isabelle/HOL mittels der Funktion *C-make-output* (Abschnitt 3.3.1).

Die Funktion $\mathcal{O}ut\mathcal{V}al_O$ stellt die punktweise Erweiterung der Funktion $\mathcal{O}ut\mathcal{V}al_p$ dar, die für einen Ausgabeport p von C seinen Wert ausgehend vom internen Zustands s_C von C nach dem Ende der Berechnungsschritte aller Teilkomponenten C_i von C ermittelt. Für Ausgabeports müssen lediglich die Fälle unterschieden werden, dass p mit einem Ausgabeport p_l einer Teilkomponente assoziiert ist, und dass p frei ist. Im ersten Fall liefert $\mathcal{O}ut\mathcal{V}al_p$ den Ausgabewert am assoziierten Ausgabeport der Teilkomponente, und im zweiten eine leere Nachricht:

$$\mathcal{O}ut\mathcal{V}al_p(s_C) \stackrel{\text{def}}{=} \begin{cases} \Pi_{p_l}(\rho_{C_i}(\Pi_i(s_C))) & \text{falls } (p_l, p) \in \mathcal{OA}_C \wedge p_l \in \mathcal{OP}_{C_i} \\ \varepsilon & \text{sonst} \end{cases} \tag{35}$$

Betrachten wir nun diese zwei Fälle bei der Funktion $\mathcal{O}ut\mathcal{V}al_O$ und der Isabelle/HOL-Funktion *C-make-output* für einen Ausgabeport p_k von C.

1) Ist der externe Ausgabeport p_k mit einem Ausgabeport p_l einer Teilkomponente C_i assoziiert, so wird der jeweilige Ausgabewert am Ende des Berechnungsschritts an den externen Ausgabeport übertragen:

$$\mathcal{O}ut\mathcal{V}al_p(s_C) = \Pi_{p_l}(\rho_{C_i}(\Pi_i(s_C)))$$

Mit $\Pi_i(s_C))$ wird der Zustand von C_i aus s_C extrahiert, mit $\rho_{C_i}(\Pi_i(s_C))$ die Ausgabe von C_i und mit $\Pi_{p_l}(\rho_{C_i}(\Pi_i(s_C)))$ die Ausgabe von p_l und somit die Ausgabe von p_k ermittelt.

In Isabelle/HOL benutzt *C-make-output* für jeden Port p_k die Ausgabeextraktionsfunktion *C-OutVal-k*, die den internen Zustand s_C von C als Parameter erhält. Ist p_k mit einem Ausgabeport p_l von C_i verbunden, sieht *C-OutVal-k* so aus:

C-OutVal-k =
get-C-i-OutPort-P-j ∘ *C-Subcomponent--C-i*

Mit *C-Subcomponent--C-i* wird der Zustand von C_i aus dem lokalen Zustand s_C extrahiert und anschließend mit *get-C-i-OutPort-P-l* der Ausgabewert von p_l ermittelt. Da der Zustand der Teilkomponente C_i und ihrer Isabelle/HOL-Darstellung *C-Comp-i* nach der Ausführung des Berechnungsschritts, wie oben gezeigt, äquivalent sind, sind auch die Werte am Ausgabeport p_k äquivalent.

2) Der Port p_k ist frei, d.h., er ist keinem Ausgabeport einer Teilkomponente C_i von C zugeordnet. In diesem Fall ist seine Ausgabe stets leer:

$$\mathcal{O}ut\mathcal{V}al_p(s_C) = \varepsilon$$

Die von *C-make-output* für p_k benutzte Ausgabeextraktionsfunktion *C-OutVal-k*, liefert in diesem Fall ebenfalls stets eine leere Nachricht:

C-OutVal-k = $\lambda x.\, \varepsilon$

Sowohl p_k als auch *C-OutPort-P-k* enthalten leere Nachricht, so dass ihre Inhalte äquivalent sind.

Somit ermittelt der dritte Teilschritt eines Zustandsübergangs – die Ausgabe – für die AUTOFOCUS-Komponente C und ihre Isabelle/HOL-Darstellung *C-Comp* äquivalente Ausgaben, d.h., für jeden Ausgabeport p_k von C und die entsprechende Isabelle/HOL-Darstellung *C-OutPort-P-k*, gilt:

$$p_k \sim \textit{C-OutPort-P-k} \tag{36}$$

Damit gilt für die Ausgabe $O = \mathcal{O}ut\mathcal{V}al_O(s_C)$ der Komponente C bzw. in Isabelle/HOL für die Ausgabe *output-new*:

$$O \sim \textit{output-new} \tag{37}$$

Die Transitionsfunktion δ_C der Komponente C konstruiert aus dem im zweiten Teilschritt berechneten lokalen Komponentenzustand und der im dritten Teilschritt ermittelten Ausgabe den neuen Gesamtzustand von C:

$$\delta_C(m,c) \;=\; \kappa_C\big(\delta_{S_C}(m,c), \mathcal{O}ut\mathcal{V}al_O(\delta_{S_C}(m,c))\big)$$

Ebenso konstruiert in Isabelle/HOL die Transitionsfunktion *C-trans-fun* den neuen Gesamtzustand von *C-Comp* aus dem neuen lokalen Komponentenzustand und der daraus ermittelten Ausgabe:

ssd-new = (|
  C-SSD-Subcomponents = *subcomps-new*,
  C-SSD-OutPorts = *output-new* |);
c-new = (| *C-Component-SSD* = *ssd-new* |)

Da sowohl die neuen lokalen Zustände (34) als auch die Ausgaben (37) äquivalent sind, sind auch die daraus bestehenden neuen Gesamtzustände von C und *C-Comp* äquivalent:

$$\delta_C(m,c) \sim \textit{C-trans-fun isa-m isa-c}$$

Damit ist die Transitionsfunktion δ_C von C und ihre Isabelle/HOL-Darstellung *C-trans-fun* äquivalent. □

Somit ist die Ausführungssemantik der Isabelle/HOL-Darstellung einer hierarchisch aufgebauten Komponente äquivalent zu ihrer AUTOFOCUS-Ausführungssemantik.

4.4 Modellsemantik

Wir wollen nun die semantische Äquivalenz vollständiger AUTOFOCUS-Modelle und ihrer Isabelle/HOL-Darstellungen zeigen.

Die semantische Schnittstelle einer AUTOFOCUS-Komponente C mit der Transitionsfunktion δ_C wird durch die Stromverarbeitungsfunktion F_C definiert, die sich als Abbildung des Berechnungszustandsstroms auf die Komponentenausgabe ergibt (vgl. [12, Abschnitt 4.1.2]):

$$F_C \;=\; \mathsf{map}(\rho_C,\, \Delta^{\omega}(\delta_C)) \tag{38}$$

Analog wird die semantische Schnittstelle der Isabelle/HOL-Darstellung *C-Comp* der Komponente C durch die Abbildung des Berechnungszustandsstroms auf die Ausgabe gegeben ((4) im Abschnitt 3.1).

Wir bezeichnen mit $Automata_C$ die Menge der Zustandsautomaten, die an den Blättern des Strukturbaums von C vorkommen:

$$\begin{aligned} Automata_C \;=\; &\{A \mid \\ &\exists n \in \mathbb{N} : \underbrace{\mathsf{SuperComponent}(\ldots \mathsf{SuperComponent}}_{n}(\mathsf{Component}(A))\ldots) = C\} \end{aligned} \tag{39}$$

Sei $\mathcal{M}$ ein AUTOFOCUS-Modell mit der Wurzelkomponente C. Seine Semantik wird durch die Stromverarbeitungsfunktion der Wurzelkomponente F_C gegeben. Die Isabelle/HOL-Darstellung *C-Comp* wird entlang des Strukturbaums der Wurzelkomponente F_C aufgebaut, wie in den Abschnitten 3.2 und 3.3 beschrieben. Für das AUTOFOCUS-Modell $\mathcal{M}$ und seine Isabelle/HOL-Darstellung *C-Comp* wollen wir nun die semantische Äquivalenz beweisen.

Zunächst zeigen wir die stärkere Behauptung, dass nicht nur die Ausgabeströme, sondern die gesamten Berechnungsströme von $\mathcal{M}$ und *C-Comp* äquivalent sind:

Theorem 1 (Äquivalenz der Berechnungszustandsströme für AUTOFOCUS-Modelle)
Seien für die Wurzelkomponente C des Modells $\mathcal{M}$ Komponentenzustände c und isa-c sowie Eingabeströme $input$ und isa-input äquivalent. Ferner seien für alle Transitionen $t_{l,i,k} \in \tau_{q_{l,i}}$ für Zustände $q_{l,i} \in Q_{A_l}$ die in QuestF formulierten Ausdrücke äquivalent zu den jeweiligen Bestandteilen in den entsprechenden Isabelle/HOL-Transitionsfunktionen A-STD-Transition-Q-l-i--Q-l-i-j-k--k. Dann sind die Berechnungszustandsströme von C in AUTOFOCUS *und Isabelle/HOL äquivalent:*

$$
\begin{aligned}
& c \sim \textit{isa-c} \wedge \mathit{input} \sim \textit{isa-input} \wedge \\
& \forall A_l \in \mathit{Automata}_C : \forall q_{l,i} \in Q_{A_l} : \\
& \forall t_{l,i,k} \in \tau_{q_{l,i}} : \mathit{ExpressionsEquiv}(t_{l,i,k},\ \textit{A-STD-Transition-Q-l-i--Q-l-i-j-k--k}) \ \Rightarrow \\
& \quad \Delta^{\omega}(\delta_C,\ \mathit{input},\ c) \sim \begin{cases} \textit{f-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \textit{if } \mathit{input} \in I^* \\ \textit{i-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \textit{otherwise} \end{cases}
\end{aligned}
$$

BEWEIS Wir beweisen die Behauptung induktiv über den Aufbau der Wurzelkomponente C: diese ist entweder Blatt des Strukturbaums des Modells $\mathcal{M}$ und besitzt eine Verhaltensverfeinerung durch einen Zustandsautomaten A, oder sie ist Knoten des Strukturbaums und besitzt eine Strukturverfeinerung durch Teilkomponenten $C_1, \ldots, C_n$ und Kommunikationsstruktur $\mathcal{CH}_C, \mathcal{IA}_C, \mathcal{OA}_C$.

Zunächt zeigen wir, dass die Transitionsfunktionen δ_C und *C-trans-fun* äquivalent sind:

$$\delta_C \sim \textit{C-trans-fun}$$

Gemäß der strukturellen Induktion über den Aufbau der Komponente C behandeln wir die beiden möglichen Fälle:

1) Verhaltensverfeinerung

   C besitzt eine Verhaltensverfeinerung durch einen Zustandsautomaten A, der die Semantik ihrer Transitionsfunktion δ_C definiert, wie im Abschnitt 3.2 beschrieben. Nach Voraussetzung sind alle QuestF-Ausdrücke in den Transitionen des Automaten A äquivalent zu ihren Isabelle/HOL-Darstellungen durch Transitionen *A-STD-Transition-Q-i--Q-j-k--k* sind. Damit ist nach dem Lemma 2 die Transitionsfunktion δ_C äquivalent zu ihrer Isabelle/HOL-Darstellung *C-trans-fun*.

2) Strukturverfeinerung

   C besitzt eine Strukturverfeinerung durch Teilkomponenten $C_1, \ldots, C_n$ und Kommunikationsstruktur $\mathcal{CH}_C, \mathcal{IA}_C, \mathcal{OA}_C$. Nach der Annahme der strukturellen Induktion sind die Transitionsfunktionen δ_{C_i} der Teilkomponenten C_i jeweils äquivalent zu ihren Isabelle/HOL-Darstellungen *C-i-trans-fun*. Nach dem Lemma 3 ist dann die Transitionsfunktion δ_C äquivalent zu ihrer Isabelle/HOL-Darstellung *C-trans-fun*.

Damit ist die Voraussetzung für das Lemma 1 erfüllt, so dass die Berechnungszustandströme von C in AUTOFOCUS und *C-Comp* in Isabelle/HOL äquivalent sind. □

Die semantische Äquivalenz der Stromverarbeitung in AUTOFOCUS und Isabelle/HOL ergibt sich als direkte Folge aus dem obigen Theorem:

Korollar 1 (Äquivalenz der Stromverarbeitungsfunktionen für AUTOFOCUS-Modelle)
Seien für die Wurzelkomponente C des Modells $\mathcal{M}$ Komponentenzustände c und isa-c sowie Eingabeströme

$input$ und isa-input äquivalent. Ferner seien Ausgabeextraktionsfunktionen ρ_C und output-fun äquivalent. Dann sind die Stromverarbeitungsfunktionen von C in AUTOFOCUS *und Isabelle/HOL äquivalent:*

$$
\begin{aligned}
&c \sim \textit{isa-c} \wedge \textit{input} \sim \textit{isa-input} \wedge \\
&\rho_C \sim \textit{output-fun} \wedge \\
&\forall A_l \in Automata_C : \forall q_{l,i} \in Q_{A_l} : \\
&\forall t_{l,i,k} \in \tau_{q_{l,i}} : \mathit{ExpressionsEquiv}(t_{l,i,k}, \textit{A-STD-Transition-Q-l-i--Q-l-i-j-k--k}) \Rightarrow \\
&\quad F_C(input, c) \sim \begin{cases} \textit{map output-fun f-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \textit{if } input \in I^* \\ \textit{output-fun} \circ \textit{i-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \textit{otherwise} \end{cases}
\end{aligned}
$$

BEWEIS Nach Theorem 1 sind die Berechnungszustandsströme von C in AUTOFOCUS und Isabelle/HOL äquivalent:

$$
\Delta^\omega(\delta_C, input, c) \sim \begin{cases} \textit{f-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \text{if } input \in I^* \\ \textit{i-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \text{otherwise (i.e., } input \in I^\infty) \end{cases}
$$

Die Stromverarbeitungsfunktion F_C ergibt sich gemäß (38) als Abbildung des Berechnungszustandsstroms auf die Komponentenausgabe durch die Ausgabeextraktionsfunktion ρ_C:

$$F_C \;=\; \mathsf{map}(\rho_C, \Delta^\omega(\delta_C))$$

Analog dazu wird in der Isabelle/HOL-Darstellung die durch *C-Comp* definierte Stromverarbeitungsfunktion als Abbildung des (endlichen oder unendlichen) Berechnungszustandsstroms auf die Ausgabe durch die Ausgabeextraktionsfunktion *output-fun* gegeben:

- Für endliche Ströme: *map output-fun* (*f-Exec-Comp-Stream C-trans-fun isa-input isa-c*)
- Für unendliche Ströme: *output-fun* ∘ *i-Exec-Comp-Stream C-trans-fun isa-input isa-c*

Da die Berechnungszustandströme $\Delta^\omega(\delta_C, input, c)$ und *f-Exec-Comp-Stream C-trans-fun isa-input isa-c* bzw. *i-Exec-Comp-Stream C-trans-fun isa-input isa-c* äquivalent sind, und die Ausgabeextraktionsfunktionen ρ_C sowie *output-fun* äquivalent sind, sind auch die Stromverarbeitungsfunktionen von C in AUTOFOCUS und Isabelle/HOL, die den funktionalen Kompositionen des Berechnungszustandsstroms mit der Ausgabeextraktionsfunktion entsprechen, äquivalent:

$$
F_C(input, c) \sim \begin{cases} \textit{map output-fun f-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \text{if } input \in I^* \\ \textit{output-fun} \circ \textit{i-Exec-Comp-Stream C-trans-fun isa-input isa-c} & \text{otherwise} \end{cases}
$$

□

5 Zusammenfassung

In diesem Bericht beschrieben wir die Darstellung stark kausaler AUTOFOCUS-Modelle in Isabelle/HOL, definierten die Semantik von STDs und bewiesen die Äquivalenz der Semantik von AUTOFOCUS-Modellen und ihren Isabelle/HOL-Darstellungen.

Die beschriebene Isabelle/HOL-Darstellung von AUTOFOCUS-Modellen soll als Basis für die Implementierung eines Übersetzers von AUTOFOCUS-Modellen in Isabelle/HOL verwendet werden.

Literatur

[1] BRAUN, P., LÖTZBEYER, H., SCHÄTZ, B., AND SLOTOSCH, O. Consistent Integration of Formal Methods. In *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000* (2000), S. Graf and M. I. Schwartzbach, Eds., vol. 1785 of *Lecture Notes in Computer Science*, Springer, pp. 48–62. 3

[2] BRAUN, P., LÖTZBEYER, H., AND SLOTOSCH, O. *Quest Users Guide*, Mar 2000. 3

[3] BROY, M. The Specification of System Components by State Transition Diagrams. Tech. Rep. TUM-I9729, Institut für Informatik, Technische Universität München, 1997. 6, 12, 19

[4] BROY, M. Service-oriented Systems Engineering: Specification and Design of Services and Layered Architectures – The JANUS Approach. vol. 195 of *Series: NATO Science Series II: Mathematics, Physics and Chemistry*. Springer Verlag, July 2005. Proceedings of the NATO Advanced Study Institute on Engineering Theories of Software Intensive Systems, Marktoberdorf, Germany, from 3 to 15 August 2004. 3

[5] BROY, M., AND STØLEN, K. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001. 3

[6] HUBER, F., AND SCHÄTZ, B. Integrated Development of Embedded Systems with AUTOFOCUS. Tech. Rep. TUMI-0701, Institut für Informatik, Technische Universität München, Dec 2001. 3, 6, 12, 19

[7] HUBER, F., SCHÄTZ, B., AND EINERT, G. Consistent Graphical Specification of Distributed Systems. In *FME '97: 4th International Symposium of Formal Methods Europe* (1997), J. Fitzgerald, C. B. Jones, and P. Lucas, Eds., vol. 1313 of *Lecture Notes in Computer Science*, Springer, pp. 122–141. 3

[8] Isabelle/HOL: Higher-Order Logic Theory in Isabelle. Part of the Isabelle distribution. 4

[9] NIPKOW, T., PAULSON, L., AND WENZEL, M. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, vol. 2283 of *LNCS*. Springer, 2002. 3

[10] NIPKOW, T., VON OHEIMB, D., AND PUSCH, C. μJava: Embedding a Programming Language in a Theorem Prover. In *Foundations of Secure Computation* (2000), F. L. Bauer and R. Steinbrüggen, Eds., vol. 175 of *NATO Science Series F: Computer and Systems Sciences*, IOS Press, pp. 117–144. 3

[11] SCHÄTZ, B., AND HUBER, F. Integrating Formal Description Techniques. In *FM'99 – Formal Methods, World Congress on Formal Methods in the Development of Computing Systems* (1999), J. M. Wing, J. Woodcock, and J. Davies, Eds., vol. 1709 of *Lecture Notes in Computer Science*, Springer, pp. 1206–1225. 3, 12, 19

[12] TRACHTENHERZ, D. *Eigenschaftsorientierte Beschreibung der logischen Architektur eingebetteter Systeme*. PhD thesis, Institut für Informatik, Technische Universität München. To appear. 1, 3, 4, 6, 10, 11, 15, 19, 20, 29, 33