Analyse von eingebetteten Echtzeitsystemen zur Laufzeit*

Thomas Reinbacher - thomas.reinbacher@gmail.com
Institut für Technische Informatik, Technische Universität Wien

Abstract: Die in dieser Kurzfassung beschriebene Dissertation stellt einen auf Hardware basierenden Ansatz für die automatisierte Analyse eingebetteter Systeme vor. Die Analyse erfolgt in Echtzeit während der operativen Phase dieser Systeme und ermöglicht es, Bugs zuverlässig zu erkennen und eine umfangreiche Fehlerdiagnose durchzuführen. Dieser Ansatz wurde erfolgreich am NASA Luftfahrzeug Swift erprobt.

Effiziente moderne Entwicklungstools erlauben es, hoch komplexe eingebettete Systeme zu entwickeln, welche zentrale Aufgaben in Komponenten der Automobilindustrie, der Luft- und Raumfahrt, etc. übernehmen. Die Verifikation dieser Systeme übersteigt dann jedoch oft die Möglichkeiten der verfügbarer Methoden [AO08]. Die Gefahr, dass Bugs während der Testphase unentdeckt bleiben, wird folglich größer [JGK+11]. Ein Beispiel für diese Gattung von Systemen ist das unbemannte Luftfahrzeug Swift [IEW10], welches am NASA Ames Research Center entwickelt wird und für irdische, wissenschaftliche Missionen [Pan12] eingesetzt wird. Abbildung 1 zeigt eine vereinfachte Darstellung, inklusive Sensoren, Aktuatoren, Flugcomputer und Kommunikationsinterface.

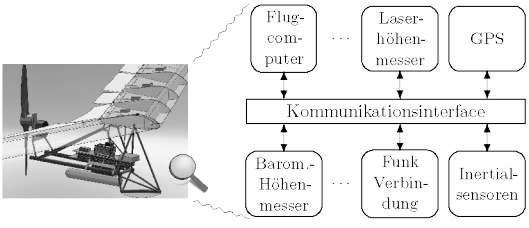

Abbildung 1: Aufbau des unbemannten Luftfahrzeuges Swift der NASA (vereinfacht).

Mit dem häufigeren Einsatz von unbemannten Luftfahrzeugen, Systemen ohne erfahrene Piloten als letzte Entscheidungsinstanz, in nationalen Lufträumen und somit dicht besiedelten Gebieten der EU [Eur12] und der USA [Dep05] muss die Frage gestellt werden, wie mit Fehlfunktionen dieser Luftfahrzeuge während des Fluges umgegangen wird. Der Verlust des Luftfahrzeugs sowie eine potentielle Gefährung von Personen am Boden sind unter allen Umständen zu verhindern. Angesichts der erwähnten Zunahme der allgemeinen Systemkomplexität beschränkt sich diese Problematik nicht nur auf den speziellen Fall der unbemannten Luftfahrzeuge, sondern stellt die Informatik vor die generelle Herausforderung, neue Ansätze zu entwickeln um mit Bugs, welche in der operativen Phase eines Systems auftreten, geeignet umzugehen. Die hier beschriebene Dissertation [Rei13] stellt einen solchen Ansatz für die automatisierte Analyse eingebetteter Systeme vor. Die Analyse erfolgt in Echtzeit während der operativen Phase dieser Systeme und ermöglicht es, Bugs zuverlässig zu erkennen und eine umfangreiche Fehlerdiagnose durchzuführen. Dieser Ansatz wurde am NASA Luftfahrzeug Swift erprobt.

*Englischer Titel der Dissertation: "Analysis of Embedded Real-Time Systems at Runtime"

Highlight 1: Anforderungsanalyse

Die Anforderungen der angestrebten Analysekomponente ergeben sich einerseits aus den Beschränkungen bei Speicher- und Rechenkapazität, Echtzeitanforderungen, usw. des zu analysierenden eingebetteten Systems [MHA+95, Men07] und andererseits aus Gesichtspunkten einer möglichst großflächigen Akzeptanz in der Industrie [JGK+11]:

UNOBTRUSIVENESS: Die Analysekomponente darf die wesentlichen Eigenschaften des zu analysierenden Systems nicht verändern. Dazu gehören die Funktionalität, Zertifizierbarkeit, temporale Eigenschaften, und sonstige Toleranzen wie Größe, Gewicht, Leistungsaufnahme, und die Telemetriebandbreite. Die Analysekomponente muss außerhalb des zu analysierenden Systems operieren und somit eine in sich geschlossene Implementierung ermöglichen.

RESPONSIVENESS: Die Analysekomponente muss das zu untersuchende System fortlaufend und regelmäßig überprüfen. Abweichungen von der Spezifikation müssen innerhalb einer engen und im Vorfeld bekannten Zeitspanne abgeschlossen sein. Nur dies ermöglicht die Einleitung von effektiven Gegenmaßnahmen um im Fehlerfall Beschädigungen am System und dessen Umwelt abzuwenden. Ein Beispiel ist die Einleitung einer koordinierten Notlandung sobald ein Ausfall des Flugcomputers des NASA Swift festgestellt wird.

REALIZABILITY: Die Analysekomponente muss nach dem plug-and-play Prinzip funktionieren und generische Schnittstellen bereitstellen um in möglichst vielen Systemen integriert werden zu können. Die unterstützte Spezifikationssprache muss einfach in einen bestehenden Arbeitsablauf integriert werden können aber auch ausdrucksstark sein, um Echtzeitanforderungen zu formulieren. Die Analysekomponente muss flexibel und ohne aufwendigen Kompilierungsprozess auf Änderungen in der Spezifikation reagieren können.

Highlight 2: Architektur zur automatisierten Analyse

Bestehende Ansätze für die Überprüfung von Abläufen in einem System bedingen meist die Instrumentierung der Hard- und/oder Software des zu prüfenden Systems und einen ressourcenintensiven Host Computer zur Ausführung des Korrektheitschecks. In der Praxis ist eine Instrumentierung jedoch schwer zu bewerkstelligen, da eingebettete Systeme oft aus einer Vielzahl von Komponenten unterschiedlichen Ursprungs bestehen. Manche sind zudem bereits zertifiziert oder stehen nur als Blackbox zur Verfügung und können daher nicht modifiziert werden. Bestehende Ansätze stehen somit im Widerspruch zu den Anforderungen bezüglich UNOBTRUSIVENESS, RESPONSIVENESS, und REALIZABILITY. Diese Anforderungen können am besten durch eine Auslegung der Analysekomponente als Hardwareeinheit bedient werden. In Anlehnung an die vorgestellten Anforderungen wird die entwickelte Analysekomponente als rtR2U2 bezeichnet (engl. für real-time, Realizable, Responsive, Unobtrusive Unit). Abbildung 2 zeigt eine überblicksmäßige Darstellung. Anstelle einer Instrumentierung baut rtR2U2 auf das passive Abgreifen aller relevanten Informationen an bereits vorhandenen Kommunikationsinterfaces auf (z.B.: industrielle Bussysteme wie CAN, FlexRay, PCI). Dies erlaubt das Nachrüsten von rtR2U2 in bestehenden Systemen und die einfache Integration in neue Designs. Durch die beabsichtigte

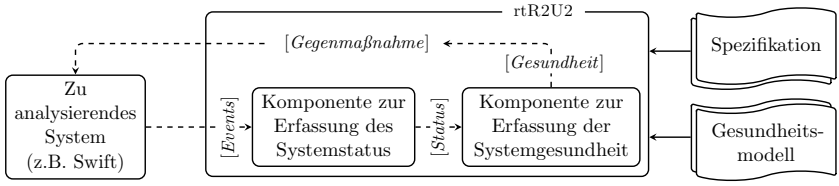

Abbildung 2: rtR2U2: Eine Analysekomponente für eingebettete Systeme.

Hardwareimplementierung von rtR2U2 muss auf Designansätze verzichtet werden die oft das Rückgrat von SW-Implementierungen darstellen: Schleifen und Rekursion (werden von CAD Tool ausgerollt und resultieren in duplizierten Hardwareinstanzen), dynamischer Speicher (Speicherplatz muss für Hardwaredesigns statisch festgelegt werden), sowie das Durchsuchen von Speicherbereichen (konstante Laufzeit kann nicht garantiert werden). Wie in Abbildung 2 dargestellt, arbeitet rtR2U2 zwei Schritten:

(i) Erfassung des Systemstatus mittels neuartigen Verifikationsalgorithmen.

Definition 1 (Systemstatus)**.** Ein Vektor $\mathfrak{s} = (\mathfrak{s}_1, \ldots, \mathfrak{s}_n)$ mit je einem Eintrag für alle n Elemente der Spezifikation. Ein Eintrag beinhaltet das Resultat der Überprüfung des zugehörigen Elements der Spezifikation. ■

Um der Anforderung REALIZABILITY gerecht zu werden, müssen formale Spezifikationen in verschiedenen Ausprägungen von temporaler Logik unterstützt werden. Diese erlauben es, temporale und logische Anforderungen an das zu analysierende System formal zu definieren. Das Herzstück dieser Komponente ist eine neuartige Familie an hoch-parallelen Algorithmen, welche den Systemstatus im Hinblick auf die vorliegende temporale Spezifikation kontinuierlich zur Laufzeit feststellen kann, siehe **Highlight 3**.

(ii) Erfassung der Systemgesundheit mittels einer neuartigen Hardwarearchitektur zur Auswertung von Bayes'schen Netzen.

Definition 2 (Systemgesundheit)**.** Ein Vektor $\mathfrak{g} = (\mathfrak{g}_1, \ldots, \mathfrak{g}_m)$ mit je einem Eintrag für die m Subsysteme des analysierten Systems. Jeder Eintrag ist eine bedingte Wahrscheinlichkeit, welche für einen Systemstatus $\mathfrak{s}$ den Gesundheitszustand eines Subsystems darstellt. ■

Diese Komponente verarbeitet die Resultate der vorgelagerten Komponente zur Erfassung des Systemstatus mit Hilfe der Theorie der Bayes'schen Statistik. Die Komponente kann Gesundheitsmodelle auswerten, welche als Bayes'sche Netze spezifiziert wurden. Das Bayes'sche Netz wird hierfür einmalig in einen Graphen übersetzt, um dann von einer hoch-parallelen Hardwareeinheit zur Laufzeit ausgewertet zu werden, siehe **Highlight 4**.

Highlight 3: Algorithmen zur Erfassung des Systemstatus

rtR2U2 verwendet eine Echtzeituhr als Zeitbasis. Mit jedem Tick $n \in \mathbb{N}_0$ dieser Uhr werden die relevanten Signale (Events in Abbildung 2) des zu analysierenden Systems abgetastet und digitalisiert. Die temporale Abfolge dieser Signale wird als Execution e bezeichnet

und ist eine unendliche Sequenz von Zuständen. Abbildung 3 zeigt die Resultate der Überprüfung des Zustandsprädikats $(\text{pitch} \geq 5°)$ anhand der Execution e. $e^n \vDash (\text{pitch} \geq 5°)$ ist immer dann erfüllt, wenn der aktuell gemessene Pitchwinkel größer oder gleich 5° ist, z.B.: für $n = 13$ gilt $e^{13} \vDash (\text{pitch} \geq 5°)$, für $n = 14$ jedoch $e^{14} \nvDash (\text{pitch} \geq 5°)$.

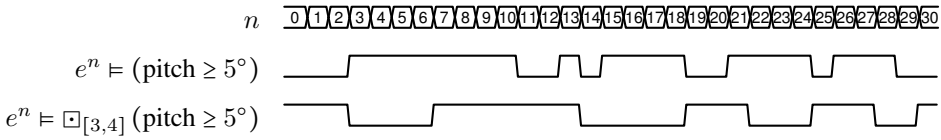

Abbildung 3: Zeitlicher Verlauf der Resultate der Überprüfung von zwei Spezifikationen.

Neben diesen einfachen Zustandsprädikaten, erlaubt rtR2U2 auch Spezifikationen in einer Vielzahl von temporalen Logiken [Pnu77] (ptLTL, ptMTL, LTL und MTL) um Echtzeiteigenschaften wie etwa harte Deadlines oder Flight Rules des Systems zu spezifizieren. Ein Beispiel ist die Eigenschaft $e^n \vDash \boxdot_{[3,4]} (\text{pitch} \geq 5°)$ die nur dann wahr ist, wenn pitch $\geq 5°$ in einem Intervall, welches in der Vergangenheit liegt und durch $J = [3, 4]$ relativ zum aktuellen Zeitstempel n beschrieben wird, wahr ist. Die Semantik dieses Operators der Logik ptMTL ist rekursiv auf einer Execution wie folgt definiert: $e^n \vDash \boxdot_J \varphi$ iff $\forall i(0 \leq i \leq n) : (\neg(n - i \in J) \vee e^i \vDash \varphi)$. Anhand dieser Semantik kann die Spezifikation $\boxdot_{[3,4]} (\text{pitch} \geq 5°)$ anhand der Execution e aus dem Beispiel in Abbildung 3 ausgewertet werden. Es lässt sich zeigen, dass $e^{21} \vDash \boxdot_{[3,4]} (\text{pitch} \geq 5°)$ (weil $e^{17} \vDash (\text{pitch} \geq 5°)$ und $e^{18} \vDash (\text{pitch} \geq 5°)$) aber $e^{22} \nvDash \boxdot_{[3,4]} (\text{pitch} \geq 5°)$. Für rtR2U2 wurden effektive Algorithmen entwickelt um solche Echtzeiteigenschaften unter den Gesichtspunkten der UNOBTRUSIVENESS, RESPONSIVENESS, und REALIZABILITY zur Laufzeit auszuwerten. Einer dieser insgesamt 13 Algorithmen wird nun kurz vorgestellt.

Algorithmus zur Auswertung des Operators $\boxdot_J \varphi$. Der Algorithmus zur Auswertung von $\boxdot_J \varphi$ in Alg. 1 benötigt eine Liste von 2-Tupel von Zeitstempeln, welche wir mit $l_{\boxdot_J \varphi}$ bezeichnen. Die Zeitstempel werden von der Echtzeituhr abgeleitet. Wir definieren das Prädikat $\textbf{valid}^{\boxdot}(T, n, J)$ mit $T \in (\mathbb{N}_0 \cup \{\infty\})^2$ wie folgt

$$\textbf{valid}^{\boxdot}(T, n, J) \equiv (T.\tau_s \leq \max(0, n - \max(J))) \wedge (T.\tau_e \geq n - \min(J)),$$

und das Prädikat $\textbf{feasible}(T, n, J)$ als

$$\textbf{feasible}(T, n, J) \equiv (T.\tau_e - T.\tau_s \geq \max(J) - \min(J)) \vee (T.\tau_s = 0 \wedge T.\tau_e \geq n - \min(J)).$$

Beispiel 1 (Algorithmus für $\boxdot_J \varphi$)**.** Wir betrachten die zeitbehaftete Spezifikation $\varphi := \boxdot_{[3,4]} (\text{pitch} \geq 5°)$ und die Execution e in obigem Beispiel. Zum Zeitpunkt $n = 0$ ist $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5°}$ leer, die Abfrage in Zeile 5 ist somit nicht erfüllt. Weil $\min(J) = 3$, liefert der Algorithmus **true** in Zeile 11 zurück. Zum Zeitpunkt $n = 3$ tritt eine ⎍⎾ Transition des Prädikats pitch $\geq 5°$ auf, und das Element $(3, \infty)$ wird zu $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5°}$ hinzugefügt (Zeile 3). Somit wird das Prädikat $\textbf{valid}^1$ in Zeile 11 wie folgt ausgewertet

$$\textbf{valid}^{\boxdot}(l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5°}[1], 3, [3, 4]) \quad = \quad (3 \leq 3 - 4) \wedge (\infty \geq 3 - 3) \quad = \quad \textbf{false},$$

[1] Wir verwenden die Notation $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5°}[i]$ um das i^{te} Element in $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5°}$ zu referenzieren.

Alg. 1 Algorithmus zur Auswertung des Operators $\boxdot_J \varphi$. Anfangsbedingung $l_{\boxdot_J \varphi} = ()$.

1: At each time $n \in \mathbb{N}_0$:
2: **if** $\_\lceil$ transition of φ occurs **then**
3: add (n, ∞) to $l_{\boxdot_J \varphi}$
4: **end if**
5: **if** $\rceil\_$ transition of φ occurs **and** $l_{\boxdot_J \varphi}$ is non-empty **then**
6: remove tail element (τ_s, ∞) from $l_{\boxdot_J \varphi}$
7: **if feasible**$((\tau_s, n-1), n, J)$ **then**
8: add $(\tau_s, n-1)$ to $l_{\boxdot_J \varphi}$
9: **end if**
10: **end if**
11: **return** $\bigvee_{k=1}^{|l_{\boxdot_J \varphi}|}$ **valid**$^{\boxdot}(l_{\boxdot_J \varphi}[k], n, J)$ in case $n \geq \min(J)$ and **true** otherwise

und wir erhalten $e^3 \not\models \psi$. Zum Zeitpunkt $n = 4$ wird Zeile 11 wie folgt ausgewertet

$$\textbf{valid}^{\boxdot}(l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}[1], 4, [3,4]) \quad = \quad (3 \leq 4-4) \wedge (\infty \geq 4-3) \quad = \quad \textbf{false},$$

und wir erhalten ebenfalls $e^4 \not\models \varphi$. Dieses Resultat wiederholt sich zu den Zeitpunkten $n \in [5,6]$. Zum Zeitpunkt $n = 7$ wird Zeile 11 wie folgt ausgewertet

$$\textbf{valid}^{\boxdot}(l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}[1], 7, [3,4]) \quad = \quad (3 \leq 7-4) \wedge (\infty \geq 7-3) \quad = \quad \textbf{true},$$

und wir erhalten somit $e^7 \models \varphi$. Durch eine ähnliche Argumentation erhalten wir $e^n \models \varphi$ für $n \in [8,10]$. Zum Zeitpunkt $n = 11$ tritt eine $\rceil\_$ Transition des Prädikats pitch $\geq 5^\circ$ auf und der Algorithmus ersetzt das Element $(3, \infty)$ in $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}$ durch $(3, 10)$ in Zeile 8. Für die Zeitpunkte $n \in [11,12]$ ist die Ausgabe des Algorithmus $e^n \models \varphi$. Zum Zeitpunkt $n = 13$ tritt eine $\_\lceil$ Transition des Prädikats pitch $\geq 5^\circ$ auf und der Algorithmus fügt das Element $(13, \infty)$ in $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}$ ein. Das Prädikat **valid**$^{\boxdot}$ ist erfüllt, daher gilt $e^{13} \models \varphi$. Zum Zeitpunkt $n = 14$ tritt eine $\rceil\_$ Transition des Prädikats pitch $\geq 5^\circ$ auf. Weil $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}$ nicht leer ist, wird das Prädikat **feasible** in Zeile 7 wie folgt ausgewertet

$$\textbf{feasible}((13,13), 14, [3,4]) \quad = \quad (13-13 \geq 4-3) \vee (13 = 0 \wedge 13 \geq 14-3) \quad = \quad \textbf{false}.$$

Das Element $(13,13)$ wird somit nicht in die $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}$ eingefügt. Das Prädikat **valid**$^{\boxdot}$ wird in Zeile 11 ausgewertet zu

$$\textbf{valid}^{\boxdot}(l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}[1], 14, [3,4]) \quad = \quad (3 \leq 14-4) \wedge (10 \geq 14-3) \quad = \quad \textbf{false},$$

und wir erhalten $e^{14} \not\models \varphi$. Zum Zeitpunkt $n = 15$ tritt eine $\_\lceil$ Transition des Prädikats pitch $\geq 5^\circ$ auf und das Element $(15, \infty)$ wird in Zeile 3 zu $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}$ hinzugefügt. Durch eine ähnliche Argumentation ergibt sich $e^n \not\models \varphi$ für die Zeitpunkte $n \in [16,18]$. Zum Zeitpunkt $n = 19$ tritt eine $\rceil\_$ Transition des Prädikats pitch $\geq 5^\circ$ auf. Das Element $(15,18)$ erfüllt das Prädikat **feasible** in Zeile 7 und somit wird $(15,18)$ zu $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}$ hinzugefügt. In Zeile 11 müssen nun zwei Elemente in $l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}$, nämlich $(3,10)$ und $(15,18)$, überprüft werden. Somit erhalten wir $e^{19} \models \psi$, weil

$$\begin{aligned}
\textbf{valid}^{\boxdot}(l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}[1], 19, [3,4]) \quad &= \quad (3 \leq 19-4) \wedge (10 \geq 19-3) \quad = \quad \textbf{false}, \\
\textbf{valid}^{\boxdot}(l_{\boxdot_{[3,4]}\,\text{pitch}\geq 5^\circ}[2], 19, [3,4]) \quad &= \quad (15 \leq 19-4) \wedge (18 \geq 19-3) \quad = \quad \textbf{true}.
\end{aligned}$$

Korrektheitsbeweise, Komplexität, und Abbildung in Hardware. In der Dissertation werden insgesamt 13 Algorithmen zur Auswertung aller Operatoren der unterstützten temporalen Logiken vorgestellt. Damit können sowohl Eigenschaften überprüft werden die in der Vergangenheit (relativ zum aktuellen Zeitpunkt) aufgetreten sind aber auch Eigenschaften welche zu einem späteren Zeitpunkt, also in der Zukunft, erwartet werden. Für alle diese Algorithmen wird der Korrektheitsbeweis angetreten, eine Untersuchung der Zeit- und Speicherkomplexität angestellt, sowie eine effiziente Abbildung in Hardware besprochen. Der oben vorgestellte Algorithmus wird bspw. so optimiert, dass in Zeile 11 nur ein Element aus $l_{\boxdot_J \varphi}$ überprüft werden muss, wodurch sich konstante Laufzeit ergibt.

Highlight 4: Bestimmung der Systemgesundheit

Die Komponente zur Erfassung der Systemgesundheit von rtR2U2 verwendet den Systemstatus (Def. 1) um Gesundheitsmodelle auszuwerten. Die Ergebnisse dieser Komponente sind bedingte Wahrscheinlichkeiten aus denen sich Gegenmaßnahmen ableiten lassen. Abbildung 4 (unten) zeigt die entwickelte Hardwarearchitektur zur massiv-parallelen Auswertung von Gesundheitsmodellen unter Berücksichtigung der Anforderungen an rtR2U2.

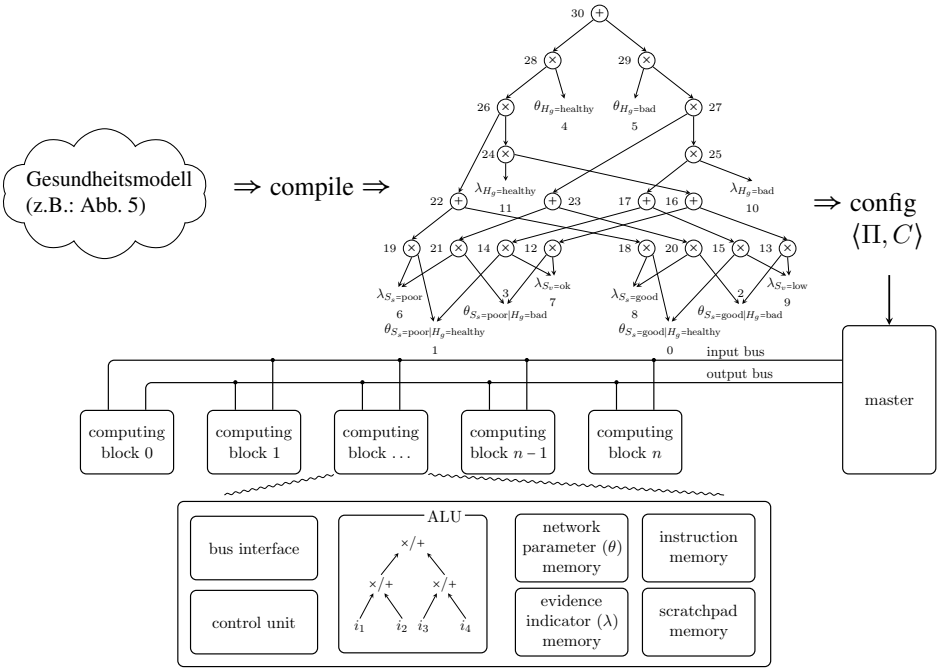

Abbildung 4: Kompiliervorgang eines Gesundheitsmodells als Bayes'sches Netz in einen Arithmetic Circuit (oben). Hoch parallele Hardwarearchitektur zur Auswertung von Gesundheitsmodellen während der operativen Phase des zu analysierenden Systems (unten).

Die Hardwarearchitektur ist darauf ausgelegt eine Graphenrepräsentation von Bayes'schen Netzen, sog. Arithmetic Circuits [Dar09], auszuwerten. Diese Graphenrepräsentation wird durch einen Kompiliervorgang aus einem beliebigen Gesundheitsmodell generiert. Aus diesem Graphen wird dann eine Instruktionsliste Π für die n parallelen Exekutionseinheiten (Computing Blocks) der Architektur generiert. Die Vorteile dieses programmierbaren Ansatzes sind: Die Hardwarearchitektur wird nur einmal auf die Zielplattform (FPGA, ASIC) synthetisiert; die Anzahl der parallelen Exekutionseinheiten ist dabei flexibel. Änderungen an Gesundheitsmodellen bedingen keine Synthese der rtR2U2 Hardware sondern können durch eine neue Instruktionsliste Π in die rtR2U2 Hardware eingespielt werden.

Highlight 5: Erprobung an einem Luftfahrzeug der NASA

Die industrielle Anwendbarkeit von rtR2U2 wird anhand der Analyse realer Flugdaten des unbemannten Luftfahrzeuges Swift (Abbildung 1) der NASA nachgewiesen. In dieser Kurzfassung wird exemplarisch eine vereinfachte temporale Spezifikation erstellt um zeitbehaftete Eigenschaften zu überprüfen, sowie ein Gesundheitsmodell entwickelt, welches die Subsysteme der Höhenmessung abdeckt. In dieser Erprobung wird nachgewiesen, dass rtR2U2, in Echtzeit und automatisch, einen Fehler in der Höhenmessung erkennt und korrekterweise eine Fehlfunktion des Laserhöhenmessers als Ursache identifiziert.

Beispiele zur temporalen Spezifikation. Für die Analyse der Flugdaten wurden temporale Spezifikationen in verschiedenen Kategorien aufgestellt, hier werden zwei genannt:

(i) Physikalische Zusammenhänge beschreiben Abhängigkeiten von Sensorgrößen, welche aus verschiedenen Subsystemen abgegriffen werden. Ein Beispiel dafür sind die Sensorgrößen Vertikalbeschleunigung und Höhe: Eine Vertikalbeschleunigung bedingt eine Änderung der Rate in der gemessenen Höhe. Dieser physikalische Zusammenhang wird durch die temporale Spezifikation φ_{R_2} erfasst.

Signale	s_vertVelocity | Vertikalbeschleunigung gemessen von Inertialsensor s_baroAltitude | Flughöhe gemessen von barometr. Höhenmesser
Spezifikation	$\varphi_{R_2} := \boxdot\,(\text{s\_vertVelocity} \geq 0.5\frac{m}{s} \rightarrow \textbf{rate}(\text{s\_baroAltitude}) \geq 0.2\frac{m}{s})$

(ii) Flight Rules sind durch nationale oder internationale Institutionen (z.B.: part 91 of the Federal Aviation Regulations in the USA [Fed13]) oder durch missionsspezifische Anforderung bestimmt. Ein Beispiel dafür ist die folgende Flugregel: Nach dem Start des Luftfahrzeuges muss ein Höhe von 600 ft nach spätestens 600 Zeiteinheiten erreicht werden. Diese Regel ist durch die temporale Spezifikation φ_{F_1} erfasst.

Signale	s_cmd | Von Bodenstation an Swift gesendeter Steuerbefehl s_laserAltitude | Flughöhe über Grund des Laserhöhenmessers
Spezifikation	$\varphi_{F_1} := \boxdot\,(\text{s\_cmd} = \text{takeoff} \rightarrow \blacklozenge_{600}\,(\text{s\_laserAltitude} \geq 600\text{ ft}))$

Ein Beispiel für Gesundheitsmodelle. Das Swift bedient sich verschiedener Subsysteme zur Bestimmung der aktuelle Flughöhe. Ein Laserhöhenmesser misst die Distanz des Luftfahrzeuges über Grund und ein Barometer gibt Auskunft über die Höhe über dem Meeresspiegel. Für den Fall, dass eines dieser Subsysteme ausfällt, oder falsche/wider-

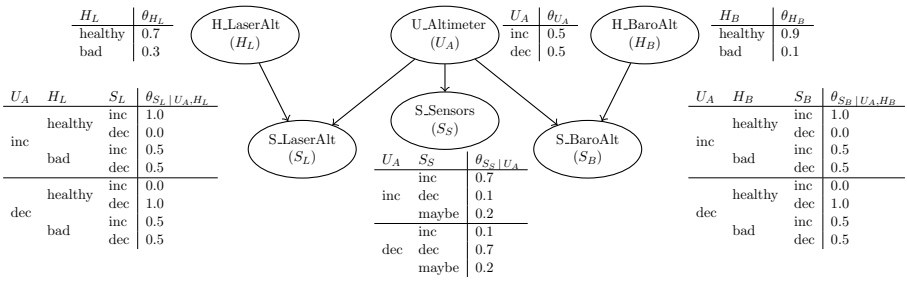

Abbildung 5: Gesundheitsmodell für die Höhenmessungssysteme des NASA Swift.

sprüchliche Informationen liefert, ist es für den Flugcomputer nicht möglich, die aktuelle Höhe genau zu bestimmen. Dies stellt eine Bedrohung der Sicherheit des Swift aber auch für dessen Umgebung dar. Darauf kann mit Hilfe von rtR2U2 regagiert werden. rtR2U2 wird dazu mit einem Gesundheitsmodell, wie es in Abbildung 5 dargestellt ist, konfiguriert.

Automatische Erkennung und Diagnose eines Ausfalls des Laserhöhenmessers. Abbildung 6 in Appendix A zeigt das Ergebnis einer industriellen, VHDL Register Transfer Level Hardwaresimulation des rtR2U2. rtR2U2 wurde hierfür mit den vorgestellten temporalen Spezifikationen und Gesundheitsmodellen (siehe Abb. 5) geladen und greift Signale mit einer Abtastfrequenz von 100 MHz von dem Kommunikationsinterface (siehe Abb. 1) des Swift ab. Die linke Abbildung zeigt die Resultate der Komponente zur Bestimmung des Systemstatus über einen gesamten Flug. Die Signalverläufe zeigen interne Signale wie zum Beispiel die Echtzeituhr (s_rtc_clk) und analoge und digitalisierte Eingangssignale von den on-board Sensoren des Swift (Höhenmessung: s_baroAltitude und s_laserAltitude, Vertikalbeschleunigung: s_vertVelocity, Pitchwinkel: s_pitch, etc.). Weiters ist der zur Laufzeit berechnete Systemstatus (z.B.: $e^n \vDash \varphi_{F_1}$ ist das Ergebnis der Überprüfung der Flight Rule φ_{F_1}) ersichtlich. Die Eingangssignale zeigen, dass während des Fluges der Laserhöhenmesser ausgefallen ist (siehe Signal: s_laserAltitude). Die rechte Abbildung zeigt einen detaillierten Ausschnitt dieser Phase des Fluges und wie rtR2U2 auf diesen Fehler reagiert. Die Signale $\Pr(H_B = \text{healthy} \mid S_L, S_B, S_S)$ und $\Pr(H_L = \text{healthy} \mid S_L, S_B, S_S)$ werden anhand des Gesundheitsmodells berechnet und repräsentieren den Gesundheitszustand der Subsysteme Barometerhöhenmesser und Laserhöhenmesser. Aus diesen Signalen zeigt sich, dass rtR2U2 diese widersprüchlichen Höhendaten unmittelbar feststellt und korrekterweise dem Laserhöhenmesser einen schlechten Gesundheitszustand attestiert, wobei der Gesundheitszustand des Barometerhöhenmessers nahezu unverändert bleibt.

Zusammenfassung

Durch die zunehmende Systemkomplexität aktueller eingebetteter Systeme steigt die Gefahr, dass (SW/HW) Bugs trotz sorgfältiger Validierungs- und Verifikationsanstrengungen nicht entdeckt werden bevor diese Ihren operativen Betrieb aufnehmen. Falls solche Bugs dann, wider Erwarten, während der operativen Phase auftreten, hat dies oft fatale Konse-

quenzen – besonders bei autonom agierenden Systemen wie etwa in der Automobilindustrie oder der Luft- und Raumfahrt. Diese Dissertation entwickelt einen neuen Ansatz welcher Bugs während der operativen Phase des Systems in Echtzeit nicht nur zuverlässig erkennen, sondern auch deren Ursache diagnostizieren kann. Dadurch können gezielte Gegenmaßnahmen eingeleitet werden um einen drohenden Systemcrash zu verhindern. Dieser Ansatz wurde erfolgreich an einem autonomen Luftfahrzeug der NASA erprobt.

Literatur

[AO08] P. Ammann und J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 1. Auflage, January 2008. ISBN: 0521880386.

[Dar09] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 1st. Auflage, 2009. ISBN: 0521884381.

[Dep05] Department of Defense, Office of the Secretary of Defense. Unmanned Aircraft Systems Roadmap 2005-2030. 2005.

[Eur12] European Commission. Towards a European strategy for the development of civil applications of Remotely Piloted Aircraft Systems RPAS. 2012.

[Fed13] Federal Aviation Administration. Federal Aviation Regulation §91, 2013.

[IEW10] C. Ippolito, P. Espinosa und A. Weston. Swift UAS: An electric UAS research platform for green aviation at NASA Ames Research Center. In *CAFE EAS IV*. April 2010.

[JGK+11] S.B. Johnson, T. Gormley, S. Kessler, C. Mott, A. Patterson-Hine, K. Reichard und J. Philip Scandura. *System Health Management: with Aerospace Applications*. Aerospace Series. John Wiley & Sons, 2011.

[Men07] O. J. Mengshoel. Designing Resource-Bounded Reasoners using Bayesian Networks: System Health Monitoring and Diagnosis. In *DX*, Seiten 330–337, 2007.

[MHA+95] D.J. Musliner, J.A. Hendler, A.K. Agrawala, E.H. Durfee, J.K. Strosnider und C. J. Paul. The challenges of real-time AI. *Computer*, 28(1):58–66, 1995.

[Pan12] M. Pandrika. Problem Solving on the Fly. *NASA Ask Magazine*, 48:30–34, 2012.

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS*, Seiten 46–57. IEEE, 1977.

[Rei13] T. Reinbacher. *Analysis of Embedded Real-Time Systems at Runtime*. Dissertation, Technische Universität Wien, Institut für Technische Informatik, 2013.

Thomas Reinbacher wurde 1984 im niederösterreichischen Hollabrunn geboren. Er studierte Elektronik und Embedded Systems an der Fachhochschule Technikum Wien und Informatikmanagement an der Technischen Universität Wien. Während seiner Studienzeit lebte er für zwei Jahre in China; dort war er in der Hardwareentwicklung für Siemens tätig und studierte an der ZheJiang Universität Mandarin. 2013 promovierte er sub auspiciis Praesidentis rei publicae (unter den Auspizien des Bundespräsidenten) am Institut für Technische Informatik an der Technischen Universität Wien. Im Zuge seiner Promotion kooperierte er als Gastwissenschaftler unter anderem mit dem Lehrstuhl Informatik 11 der RWTH Aachen sowie der Robust Software Engineering Group des NASA Ames Research Centers und veröffentlichte 21 peer-reviewed Beiträge auf Konferenzen und in drei Journals, drei seiner Arbeiten wurden mit einem Best-Paper Award ausgezeichnet.

A Appendix: Hardwaresimulation

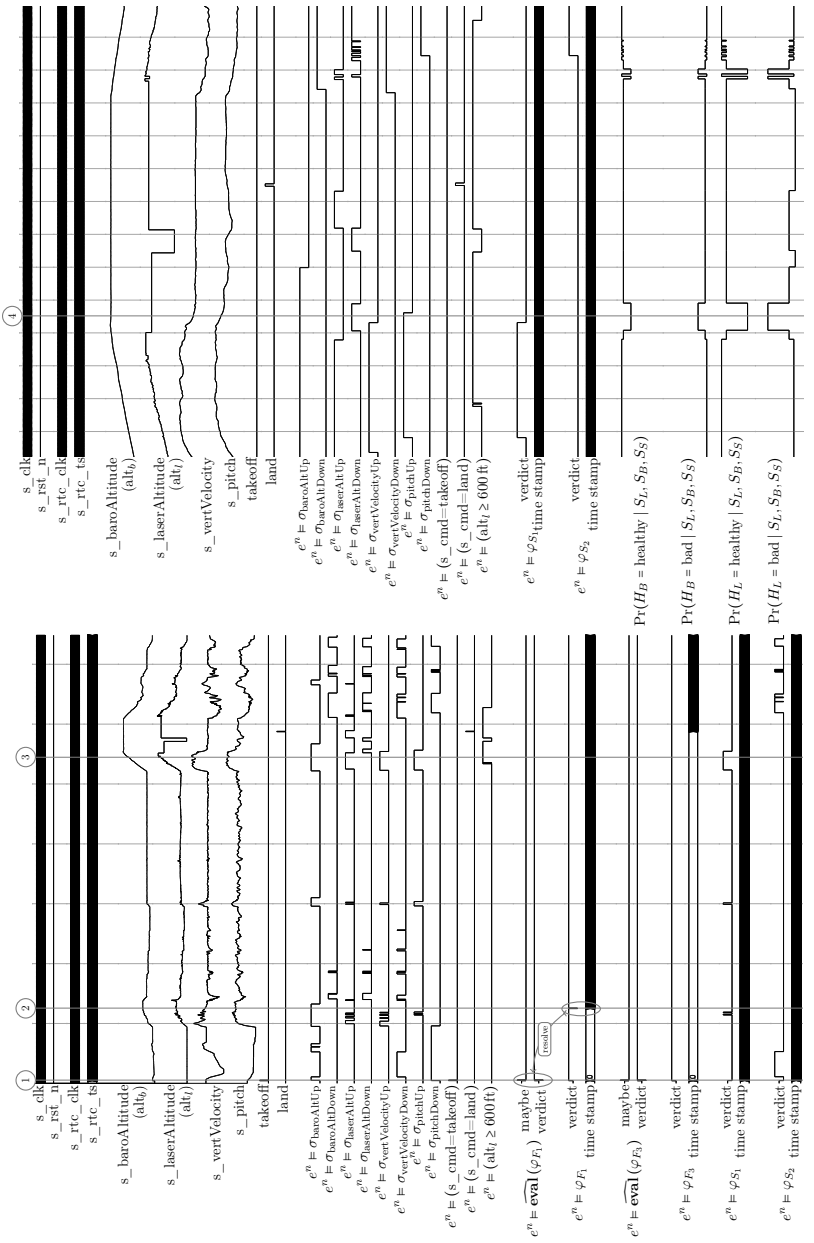

Abbildung 6: Signalverläufe einer low-level Hardwaresimulation des rtR2U2.