The 32nd Annual Conference of the Japanese Society for Artificial Intelligence, 2018

順序入れ替えテストケースの蓄積を実現するテスト実行環境

An Testing Environment to Accumulate Test Cases for Execution Order Variation

青山裕介 *1　黒岩丈瑠 *1　久代紀之 *1
Yusuke Aoyama　Takeru Kuroiwa　Noriyuki Kushiro

*1 九州工業大学
Kyushu Institute of Technology

Many kind of Internet of Things (IoT) devices are recently developed. The IoT devices connect to an open platform and communicate each other with message exchange. Since the order of the message exchange is not deterministic, developers tests IoT devices in exhaustive order. The tests are usually conducted manually, which brings two issues: p1) testers possibly miss some of the order; p2) testers possibly overlook some evaluation items.

As a solution, we developed a testing environment, which has a model checker that communicates with software on actual devices or an emulator. The testing environment solves p1) by automatic evaluation. The testing environment solves also p2) by automatic and simultaneous evaluation of evaluation items expressed as a Linear Temporal Logic (LTL) expression by integrating multiple LTL expressions with and-operator.

We applied the test execution environment to three market defects and confirmed that the test execution environment tested the defects correctly.

1. はじめに

今日，オープンなプラットフォームに接続し，複数製品を協調動作させる Internet of Things (IoT) 製品の開発が広く行われている．この IoT 製品には次の特徴がある．

1. IoT 製品がマルチベンダによって開発されること
2. IoT 製品間でのメッセージ交換が非決定的な順序で発生すること

上記の特徴 1, 2 から，製品評価において，開発した IoT 製品のメッセージの交換が順序の仮定なく動作すること，すなわち，あらゆる順序でメッセージ交換を発生させても問題が発生しないことを確認する必要がある．

一方でこうしたあらゆる順序を網羅するようなテストケースは膨大な数になってしまう．このため，現実的には例えば障害発生のリスクに鑑みてテストする対象を絞り込む方法 [Craig 02] などにより，限定的な種類のメッセージ交換について順序を入れ替えたテストを実施する．

現状こうしたメッセージ交換順序を入れ替えるテスト（以降，順序入れ替えテストと呼ぶ）はテスト技術者が手動で実施するために，絞り込みを実施した後も次の課題が存在する．

p1. 順序入れ替えテストケースの操作手順の網羅ミス

p2. 複数ある評価項目の評価の見落とし

このため，テストケースを蓄積して製品シリーズに渡って継続的に不具合を検出していくことが難しく，検出できるはずの不具合をテストをすり抜けて流出させ，過去不具合が再発してしまう可能性があった．

本研究では課題 p1, p2 を解決するために，

s1. 順序入れ替えテストケースの自動生成・実行

s2. 複数評価項目の自動評価

を実現することで順序入れ替えテストを自動化するテスト実行環境を開発した．また，s1 を個別のメッセージ交換のテストケースを元に行うことで，継続的な順序入れ替えテストのためのテストケースの蓄積を実現した．

2. 提案するテスト実行環境

節 1. の s1, s2 実現のためのテスト実行環境は図 1 のような構成となっている．本研究に先立ち開発したテスト実行環境 [Kuroiwa 16] を用いることで，s1 を実現し，さらに [Kuroiwa 16] を拡張することで節 1. の p2 の実現を図る．

2.1 拡張前のテスト実行環境

[Kuroiwa 16] は図 1 ③，⑤ を搭載し，③ ↔ ⑤ ↔ ⑥ の間でメッセージ交換を行う．このメッセージ交換をモデル検査器 Spin [Holzmann 04] を拡張したものにより行うことで，実装されたソフトウェアの順序入れ替えテストの自動実行を実現している．

Spin は従来は設計の評価に用いられてきたツールであり，入力言語として Promela を扱う．Promela 中でプロセスごとのシーケンス（本論文においては個別のメッセージ交換のシーケンス）を定義して Spin に入力すると，シーケンス同士の順序の入れ替えを自動的に網羅して実行することができる．

[Kuroiwa 16] はこの Spin の自動的かつ網羅的なテスト実行機能を実装されたソフトウェアに適用できるように，Spin に次のような拡張を施している．(詳細は [Kuroiwa 16] を参照されたい)．

1. PC と実装されたソフトウェアを搭載する組み込み機器との間の通信装置（図 1⑤）の開発
2. 実装されたソフトウェアの状態初期化機構の Spin への追加

2.2 テスト実行環境拡張内容

節 1. s2 は，Linear Temporal Logic (LTL) [Clarke 99] を使って実装済みのソフトウェアについての評価項目の記述・評価の実施を可能にすることで実現する．

以降の小節にて，まず LTL を使って節 1. の p2 を実現する評価項目記述方法について，続いて LTL で記述した評価項目の評価を実施する方法について述べる．

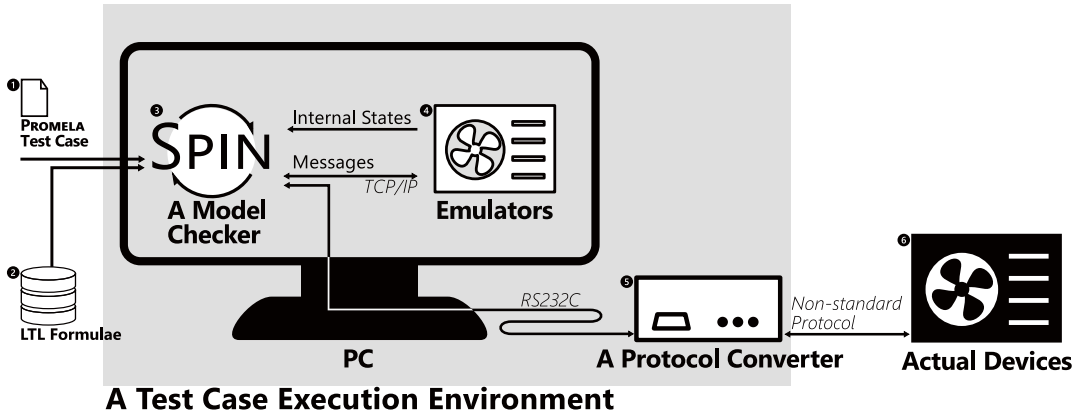

図 1: テスト実行環境の構成

2.2.1 LTL による評価項目の記述

順序入れ替えテストの評価では，組み合わせた個々のメッセージ交換シーケンスがどの順序で起こるかを一つに決定することはできない．しかし，個々のメッセージ交換を評価するために，非決定的ながらもテストの実行シーケンスのある時点を指定し，「いつ」評価を実施すればよいのかを評価項目中で表現する必要がある．

このような「いつ」を表現した評価項目の記述のために，本研究では評価項目の記述に LTL を採用した．

LTL は「いつか（記号: ◇）」や「常に（記号: □）」といった時系列情報を扱うことができる論理であり，Spin は標準で評価項目として扱うことができるものである．この LTL の採用により，メッセージ交換（受信 → 応答）のシーケンスの最終結果として「いつか」期待する状態になることの確認（例:「常に，応答実施後（P）にはある状態（S）である（$\Box(P \rightarrow \Diamond S)$）」）のための，あるいは複数のメッセージ交換の組み合わせが問題を起こさないことの確認（例:「ソフトウェアの内部状態が望まない状態になる (S) ことはずっとない ($\Box\neg S$)」）のための評価項目を表現可能となる．

本研究と同じくモデル検査器を用いて実装済みのソフトウェアをテストする手法 [Vries 00, Tretmans 03, Larsen 05] と比較すると，LTL 式を用いたことで，テスト判定において利点がある．[Vries 00, Tretmans 03, Larsen 05] ではテスト結果の判定として，ソフトウェアへの入力に対する出力が妥当かという入出力の関係を用いる．一方，本研究では LTL 式により，ソフトウェアの内部状態の状態変化によってテスト結果を判定する．このため個別の入出力の関係に加え，テストシーケンス全体に渡った異常状態の不発生，というような入出力に直接紐付かない評価も可能になっている．

また LTL 式は論理式であるので，個別のメッセージ交換の評価項目を論理演算子`&&`(And) で結合することで，複数の評価項目をまとめて確認する評価項目が表現可能である．

個別のメッセージ交換についての評価項目と，過去不具合事例から得られた評価項目を図 1 ② に蓄積しておくことで，順序入れ替えテスト実施時に組み合わせ，まとめて評価することができる．

2.2.2 LTL で記述した評価項目の評価の実施

節 1. s2 の評価の実行を実現するために，エミュレータ [Kuroiwa 15] を使ったソフトウェアの内部状態の取得機能を [Kuroiwa 16] に追加した．

エミュレータ [Kuroiwa 15] は API としてエミュレータ上で動作しているソフトウェアの内部状態を取得する機能を提供している．このエミュレータから取得したソフトウェア内部状態を期待値と比較する LTL 式を記述することで，Spin によって自動で評価し，節 1. s2 を実現できる．

表 1: ビル用空調システムを構成する機器

機種	1 年あたりのリリース台数
室内機	約 50 台
室外機	約 50 台
リモートコントローラ	約 5 台
システムコントローラ	約 1–2 台

LTL 式中での内部状態の比較自体は Spin の標準機能で実現できる．Spin の入力となるテストケース記述用の言語 Promela には，任意の C 言語プログラムを埋め込む構文 `c_expr` というものがある．構文 `c_expr` は，`c_expr` 直後に続く{}の内部に埋め込んだ C 言語の式を評価し，評価結果を真偽値として扱うことができる．`c_expr` は LTL 式を記述する構文の中にも書けるので，C 言語プログラムとしてエミュレータから取得したソフトウェア内部状態と期待値を比較する式を書くことで，節 1. の s2 を実現できる．

3. 実験

3.1 実験対象システム

本論文の評価のために提案手法を適用するビル用空調システムについて説明する．

適用対象のビル用空調システムは表 1 のような複数機器が含まれる．機器は同じネットワーク上でメッセージ交換を行いながら動作しており，ネットワークを流れるメッセージの到達順序は保証されていない．危機感のメッセージ交換順序に仮定が置けないことに鑑みると，各機器をそれぞれ一つの IoT 製品とみなすことができる．

表 1 のように毎年複数の機種がリリースされ，システムのネットワーク上に接続する機器は，リリースされた年次を問わず任意に組み合わせることを許容している．このため，接続される機器の違いによらず，任意の順序で発生するメッセージ交換が問題なく動作することを評価する，順序入れ替えテストが必要なシステムである．

表 2: 不具合事例概要

No.	概要
1	**単一機能のメッセージ交換順序の誤った仮定:** メッセージの到達順序の保証されないネットワーク上で順序の仮定を置いてしまっていた．この結果，ある複数メッセージの送付を必要とする機能が定められた順序でメッセージを受信しないと作動しなかった．
2	**機能の競合:** 機能 A と機能 B が同時に実行されると，機能 B が変更する状態によって，本来影響を受けるはずのない機能 A の振る舞いが変わった．
3	**モードの違う機器の競合:** 機器 X のモードに応じて機器 B の状態を変える機能について，モードの異なる機器 A と機器 C が共通の機器 B に本機能を実行した結果，制御される状態が一定しなかった．

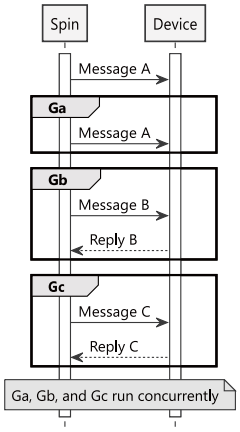

図 2: 表 2 の No. 1 のテストシーケンス

表 3: 表 2 の不具合が再発していないことを確認する LTL 式

No.	LTL 式
1	$\Diamond(StateS == value)$
2	$\Diamond(StateC' == StateC)$
3	$\Box(StateA! = StateT)$

表 4: テストの結果

No.	結果	期待結果
1	事例発生せず	事例発生せず
2	事例発生	事例発生
3	事例発生せず	事例発生せず

3.2 実験内容

本研究で開発したテスト実行環境を用いて，LTL 式を使った自動的な評価が実施可能であることを確認する．このために，節 3.1 のビル用空調システムに対し，後工程へ不具合が流出した表 2 の 3 件の事例を本テスト実行環境でテストする．

確認に用いるテストケースの操作手順としては，表 2 の不具合事例を引き起こした図 2–4 のテストシーケンスを用いる．検証に用いる LTL 式は，不具合事例が再発していないことを確認する表 3 を使用する．

4. 評価結果

テストの実行について，対象の機器には，表 2 の No. 1 と No. 3 については不具合を修正済みのものを用い，No. 2 については不具合を修正する以前の機器を用いた．テストの実行の結果実行の結果，表 4 の成否判定結果を得た．

5. 考察

表 4 の結果を見ると，不具合を残した機器で実施した表 2 の No. 2 の事例については，意図した通り不具合の発生を検出したことが確認できる．一方，不具合を修正済みの機器に対して実施した表 2 の No. 1, No. 3 の事例については，意図通りテストがパスしたことが確認できる．これは本テスト実行環境により偽陽性も偽陰性なく評価が実現できたことを示している．ここから，テスト実行環境によって，LTL 式で記述した評価項目を用いた自動的評価が可能であることを確認できた．

また，メッセージの送信順序の網羅について，表 2 の No. 1 のテストは，$3! = 6$ 通り，表 2 の No. 2 のテストは，$7!/6! = 7$ 通り，表 2 の No. 3 のテストは，$2! = 2$ 通りと，いずれの事例でも起こりうる全ての順序でメッセージを送信するテストが実施されることを確認した．この内，表 2 の No. 1 の事例から 2 例を挙げると図 5, 図 6 のようになっており，図 2 で並行に動作すると示されている Message B，Message C の送信順序が入れ替わっていることが確認できる．この結果から，本テスト実行環境により節 1. の s1 が実現できていることが確認された．

6. まとめと今後

本研究では，機器間のメッセージ交換の順序を仮定できない IoT 機器について，任意の順序のメッセージ交換をテストするテスト実行環境を開発した．本テスト実行環境では，従来の人手によるメッセージ交換順序の入れ替えテストの持つ次の 2 つ課題:

p1. 順序入れ替えテストケースの操作手順の網羅ミス

p2. 複数ある評価項目の評価の見落とし

を防ぐために，LTL 式を用いた実装済みのソフトウェアの自動評価を行うことで

s1. 順序入れ替えテストケースの自動生成・実行

s2. 複数評価項目の自動評価

の実現を図った．この LTL 式を用いた実装済みのソフトウェアの自動評価が実現できていることを確認するため，3 件の不具合事例をテスト実行環境でテストした．この結果，開発したテスト実行環境は，適用対象の過去不具合事例を自動的かつ正しく評価可能であることを確認した．

今後は s1，s2 が達成されたことを確認するために更に評価を充実させる予定である．

謝辞

本研究は，JSPS 科研費 16K00100，16H01836，JST CREST の支援を受けたものである．

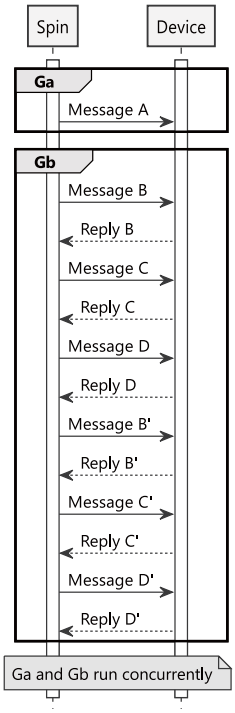

図 3: 表 2 の No. 2 のテストシーケンス

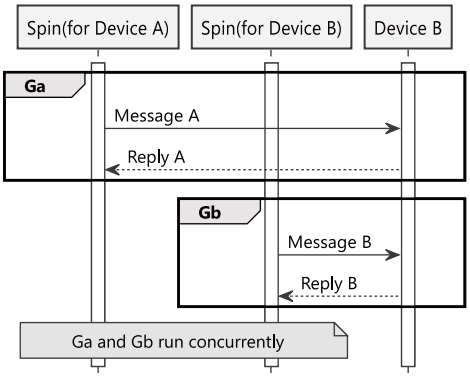

図 4: 表 2 の No. 3 のテストシーケンス

参考文献

[Clarke 99] Clarke, E. M., Grumberg, O., and Peled, D.: *Model checking*, pp. 31–33, MIT press (1999)

[Craig 02] Craig, R. D. and Jaskiel, S. P.: *Systematic software testing*, pp. 17–34, Artech House (2002), 宗 雅彦（監訳） and 成田 光彰（訳）, 体系的ソフトウェアテスト入門, 日経 BP 社, 2004

[Holzmann 04] Holzmann, G. J.: *The SPIN model checker: Primer and reference manual*, Vol. 1003, Addison-Wesley Reading (2004)

[Kuroiwa 15] Kuroiwa, T. and Kushiro, N.: Testing environment for embedded software product lines, in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–7IEEE (2015)

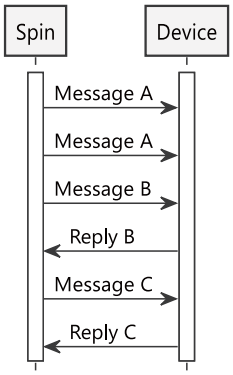

図 5: 表 2 の No.1 の実行結果の例 1

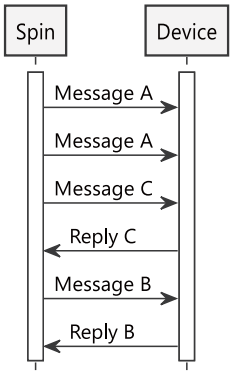

図 6: 表 2 の No.1 の実行結果の例 2

[Kuroiwa 16] Kuroiwa, T., Aoyama, Y., and Kushiro, N.: Testing Environment for CPS by Cooperating Model Checking with Execution Testing, *Procedia Computer Science*, Vol. 96, pp. 1341–1350 (2016)

[Larsen 05] Larsen, K. G., Mikucionis, M., Nielsen, B., and Skou, A.: Testing real-time embedded software using UPPAAL-TRON: an industrial case study, in *Proceedings of the 5th ACM international conference on Embedded software*, pp. 299–306ACM (2005)

[Tretmans 03] Tretmans, J. and Brinksma, E.: TorX: Automated Model-Based Testing, in Hartman, A. and Dussa-Ziegler, K. eds., *First European Conference on Model-Driven Software Engineering*, pp. 31–43 (2003)

[Vries 00] Vries, de R. G. and Tretmans, J.: On-the-fly conformance testing using SPIN, *International Journal on Software Tools for Technology Transfer (STTT)*, Vol. 2, No. 4, pp. 382–393 (2000)