1U-08

無線センサネットワークにおけるデッドロック検出のための形式手法

池田　愛大†　　秋山　直輝‡　　宮崎　敏明†

会津大学コンピュータ理工学部†　　会津大学大学院コンピュータ理工学研究科‡

1. はじめに

無線センサネットワークは、環境モニタリングに有効である。しかし、各センサノードの振る舞い、または互いに影響を及ぼし合うセンサネットワークとしての振る舞いを検証し、デッドロックを事前に見つけることは難しい。我々は、Communicating Sequential Processes (CSP) に基づく形式検証ツールと、無線センサノードで実際に動作する記述を同時に生成するシステムを実現した[4]。本稿では、周期呼び出しのある実用的な例に対し、我々のシステムを適用した結果を示す。

2. 無線センサネットワークに対する形式検証

無線センサネットワーク(WSN)において、各無線センサノード(以下、単にノード)は自身にプログラムされた動作を実行する。そして各ノードが互いに影響を及ぼし合い、全体として WSN としての機能を果たす。よって、個々のノードの動作を解析するだけでは WSN 全体におけるデッドロックの発生の有無を確認することは難しい。

我々が提案するシステムの概要を図 1 に示す。本システムは Funclet[2]言語を拡張した Funclet+という言語とその言語変換器からなる。Funclet+は、プロセス代数の 1 つである CSP[1]に基づく言語である。ユーザが Funclet+を用いてノードの動作定義を行うと、言語変換器は当該 Funclet+記述から形式検証用コード(RTS)とC言語記述を生成する。形式検証用コードは、形式検証ツールである Process Analysis Toolkit (PAT)[3]に入力され、形式検証される。一方、C 言語記述はそのままコンパイラに入力され、対象ノード上で実行できる実機用コードが得られる。すなわち、我々のシステムを用いることにより、検証用と実機用のコードを別々に書く必要がなく、1 つの動作仕様から形式検証と実機用コードを生成でき、検証した動作を実行する記述を書く際に生じる人的な誤りを避けることができる。また、形式検証手法を用いることにより、シミュレータやテストベンチデータを用いることなく動作検証を行うことができる。我々は、本システムを用いてノード同士が通信する際に起こる典型的なデッドロックを検出できることを示した[4]。しかし、WSN の実用的なアプリケーションではセンサデータの取得、周囲との通信などを周期的に行うのが一般的であり、そのような動作を含む例を検証しなければシステムの有用性を十分に示しているとは言えない。今回は機能の周期呼び出しのある実用的な例として彩色問題を取り上げ、デッドロックが形式検証により検出できることを示す。

3. 彩色問題におけるデッドロック検出

3.1. 彩色問題の概要

例として、温度センサは赤、加速度センサは黄といったように、各ノードで動作させるべきセンサを種別ごとに別の色を割り振る。いま、環境条件によって、動的に各センサをどのノードに割り当てるかを決定する問題は、彩色問題として定式化できる。ここでは、ノードの総数が n の WSN を考える。図 2 のように各ノードは、赤と黄の 2 つの状態を持つ。近傍のノードと自身の状態を確認して、n が偶数ならば、n/2 が赤、残る n/2 が黄となるように色を割り当てる。また、n が奇数ならば、赤の総数または黄の総数が他方より 1 つ多くなるように各ノードの色を割り当てる。各ノードには事前に連続的な番号が振ってあり 1 番目のノードから n 番目のノードまで、自身の状態(色)を知らせた後に、状態数の集計とその結果から自身の状態を変更するかどうかの判定を行う。さらに、自身の状態を変更した時のみ、他ノードの状態変更を促し、より早く全体の安定を図るために、自身の状態をブロードキャストする。

3.2. デッドロックとなる部分

上述した彩色問題の Funclet+記述を付録 1 に示す。スペースの都合上、一部を抜粋しコメントで補完している。ここで、ノード番号を指定する際、付録 1 の 10 行目の if 文において、<とすべきところを≦と書き間違えて、ノードの最大番号より 1 つ大きなノード番号を指定することで存在しない番号のノードにもアクセスするプログラムを書いたと仮定する。本記述ミスがデッドロック検出と、実機無線センサノードの振る舞いにどのような影響を及ぼすかを検証する。

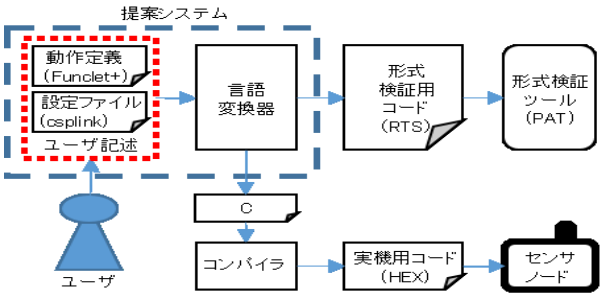

図 1　システムの全体図

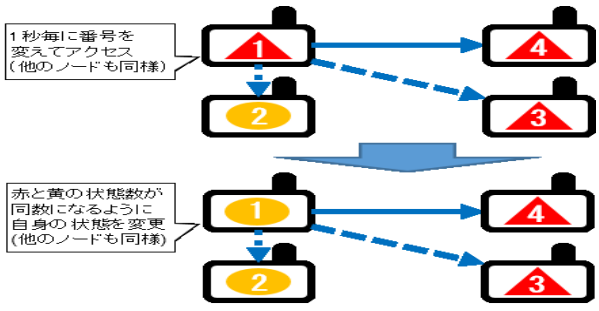

図 2　彩色問題

4. 形式検証とセンサノードによる実験

4.1. Funclet+の変換結果

コメント及び改行を除いた「記述ミス無し」の Funclet+記述の変換結果は表 1 のとおりである。また、Funclet+記述及び生成されたコード中の<と≦が変わるだけで「記述ミス有り」の場合も同じ行数である。表 1 中の”Funclet+”はFunclet+記述である。”csplink”は、想定するノード数などの検証条件の設定ファイルであり、ユーザが記述し Funclet+記述とともに言語変換器に入力される。”RTS”は PAT に入力される形式検証用コード、”C”は無線センサノードで動作するC言語記述であり、言語変換器により自動生成されたものである。

Formal Approach to Detecting Deadlocks in Wireless Sensor Networks
†Akihiro Ikeda, ‡Naoki Akiyama, †Toshiaki Miyazaki
†School of Computer Science and Engineering, The University of Aizu
‡Graduate School of Computer Science and Engineering, The University of Aizu

Copyright ©2017 Information Processing Society of Japan.
All Rights Reserved.

表 1 各ファイルの行数(コメント、改行を除く)

ユーザ記述	Funclet+	csplink	合計
	66	24	90
自動生成	RTS	C	合計
	112	105	217

4.2. 形式検証によるデッドロック検出

表 2 の環境で、ノード指定の部分に「記述ミス無し」の彩色問題とノード指定に「記述ミス有り」の彩色問題、それぞれの Funclet+記述から自動生成された形式検証用コード(RTS)を、PAT により形式検証した。ノード数 n は、n=2 で検証した。結果を表 3 に示す。「記述ミス有り」の場合、デッドロックが直ちに検出された。

表 2 検証環境

OS	Windows10 Home 64bit
検証ツール	PAT Version 3.5.1
CPU	Intel® Core™ i5-4460 CPU @ 3.2GHz
メモリ	16GB

4.3. 実機無線センサノードによる振る舞いの確認

実機無線センサノードで実験する際は、ノード数 n=4 を想定するように付録 1 の 10 行目を変更した。言語変換器により自動生成された C 言語記述を 920MHz 帯で通信する独自開発した無線センサノードに実装し、実際に、彩色問題の記述ミスの有無により収束時間がどのように変化するか測定した。振る舞いの確認はノードが持つ赤と黄の LED の点灯によって行い、最後に電源を入れたノードの LED が発光した瞬間からすべてのノードの LED の色が変化しなくなるまでの時間を計測し、その後 20 秒どのノードの色も変化しなければ計測した時間を収束時間とした。ノード内の動作は 1 秒周期で実行されるように定義し、各ノードの動作が順次実行されるとして、4 つのノードに 5 周期分アクセスするのにかかる最大時間は 20 秒となる。ここでは、全てのノードの状態が変化しなくなってから、その後 20 秒間どのノードの状態も変化しなければ、WSN 全体が安定したものとみなした。表 4 に実験結果を示す。同一条件で、3 回試行した。実機では「記述ミス無し」の方が平均で 0.7 秒早く収束した。

表 4 実機での彩色問題の収束時間(単位:s)

	1回目	2回目	3回目	平均
記述ミス有り	3.4	3.5	3.5	3.5
記述ミス無し	2.7	2.8	2.9	2.8

5. おわりに

周期呼び出しのある実用的なアプリケーションである彩色問題を Funclet+を用いて記述し、提案システムを用いて自動生成された形式検証用コードを形式検証ツール PAT に入力し、「記述ミス有り」のプログラムからデッドロックを検出できることを確認した。また、同時に生成された C 言語記述を実機無線センサノードに実装し、PAT により検出されたデッドロックが実際の WSN に及ぼす影響も確認した。今回の彩色問題の実験では、「記述ミス無し」のデッドロックが検出されないプログラムの方が、システム全体が安定するまでに要する時間が平均 0.7 秒短かった。これは WSN の振る舞いとしてより安定していると言える。

現状の実装では自動生成されるコード、特に形式検証用コードは、プロセスを細かく分け過ぎているため、検証時に状態爆発が起こり、メモリ不足により検証できなくなる場合があり、コードの自動生成法の改善が必要である。また、Funclet+言語自体も、動作仕様の定義がより容易にできるように改善していきたい。

謝辞

本研究の一部は、科研費(15K00133)により実施された。

参考文献

[1] C.A.R. Hoare, “Communicating Sequential Processes,” Communications of the ACM, vol. 21, no.8, pp. 666-677, August 1978, DOI:10.1145/359576.359585

[2] S. Jaskó, and G. Simon, “CSP-Based Sensor Network Architecture for Reconfigurable Measurement Systems,” IEEE Transactions on Instrumentation and Measurement, vol. 60, no.6, pp. 2104-2117, May.2011

[3] Process Analysis Toolkit, http://pat.comp.nus.edu.sg/

[4] N. Akiyama, A. Ikeda and T. Miyazaki, "Deadlock-free Behavior Definition for Wireless Sensor Nodes Using Formal Verification, " IEEE student session in 2016 Tohoku-Section Joint Convention of Institutes of Electrical and Information Engineers, Japan (平成 28 年度電気関係学会東北支部連合大会), 2A08, Sendai, Aug. 2016


```
   //電源 ON 時に 1 回だけ呼ばれる
1: activate,
2:     n = 2  //ノード数 n が 2 である
3:     nomal!TIMER_CTRL 1, //funclet 名 nomal を 1 秒周期で呼ぶ
4:     address = 0,
5:     nodes_color[SELF] = RED,
6:     _LedRed!1,    //LED は最初赤く光る。検証対象外で実機用
7:     pass

8: nomal?,
       //各ノードの funclet 名 access にシーケンシャルにアクセス
9:     access!_A_node_0+address nodes_color[SELF],
10:    if(address < n-1){ //ノード数の最大値チェック
11:        address++
12:    } else {  //最終の番号まで到達したら集計と自身の状態変更の判定へ
13:        address = 0,
14:        redCnt = 0,
15:        yellowCnt = 0,
16:        for(j:_N_node){  //自身を含めすべてのノードの色をチェック
17:            if(nodes_color[_A_node_0+j] == RED){
18:                redCnt++
19:            },
20:            if(nodes_color[_A_node_0+j] == YELLOW){
21:                yellowCnt++
22:            }
23:        },
           //色数を計算し、最小のものと最大のものを把握する
24:        min = redCnt,
25:        max = redCnt,
26:        maxColor = RED,
27:        minColor = RED,
28:        if(yellowCnt <= min){
29:            min = yellowCnt,
30:            minColor = YELLOW
31:        },
32:        if(yellowCnt >= max){
33:            max = yellowCnt,
34:            maxColor = YELLOW
35:        },
           //最大ノード数の色と最小のノード数の色の差が 0 か 1 の時、または自
           身のノードが最大のノード数の色以外の場合自身の状態の変更なし
36:        if(min != max && min + 1 != max && nodes_color[SELF] == maxColor){
37:            //最小のノード数の色に自身の状態を変更
38:            if(minColor == RED){
39:                nodes_color[SELF] = RED,
40:                _LedYellow!0, _LedRed!1    //  黄 LED を消灯、赤 LED を点灯
41:            },
42:            if(minColor == YELLOW){
43:                nodes_color[SELF] = YELLOW,
44:                _LedYellow!1, _LedRed!0    //  黄 LED を点灯、赤 LED を消灯
45:            }, //状態を変更したとき状態の安定のため周囲にブロードキャスト
46:            access!BROADCAST nodes_color[SELF]
47:        }
48:    },
49:    pass
   //周囲から送られてきた色の情報を自身の配列に格納する
50:access?from x,
51:    nodes_color[from] = x,
52:    pass
```

付録 1 Funclet+による彩色問題の記述例

表 3 PAT による形式検証結果

	ノード数	Verification Result	Visited States	Total Transitions	Time Used (ms)	Estimated Memory Used (MB)
記述ミス有り	2	The Assertion (system() deadlockfree) is NOT valid.	81	80	17	11
記述ミス無し	2	The Assertion (system() deadlockfree) is VALID.	$2.5\text{x}10^5$	$7.4\text{x}10^5$	$1.1\text{x}10^4$	78

Copyright ©2017 Information Processing Society of Japan.
All Rights Reserved.