Vol.2016-IOT-32 No.38
2016/3/4

NetCore プログラムの Coq による検証

伊達 弘明[†1,a)]　吉浦 紀晃[†1,b)]

概要：柔軟なパケット転送を行いたいという要求により，OpenFlow が提案されている．OpenFlow スイッチの導入により大規模なネットワーク構築や構成変更をより柔軟に行えるようになった一方，OpenFlow コントローラの転送ルールをプログラミングするため，NetCore，Trema，Frenetic など様々な言語やソフトウェアが開発されてきた．これらのうち，NetCore とは関数型言語 Haskell でプログラミングできる言語であり，宣言型であること，モジュール構造を持っているなどの特徴がある．NetCore に対する研究として，定理証明支援ソフト Coq 上において，そのプログラムの健全性と完全性を証明する事が行われた．この研究により，NetCore によりプログラミングされた言語が，プログラマの意図した通りにパケットを転送できるかを検証する事が可能となった．本稿では，イッチ同士の接続による閉路が存在するとき，NetCore による適切なルール設定により論理的にループを除去することが可能であるかを Coq を用いて検証した．

Verification of NetCore Programs by Coq

Hiroaki Date[†1,a)]　Yoshiura Noriaki[†1,b)]

Abstract: OpenFlow is proposed for flexible forwarding of packets. OpenFlow enables to construct and change large networks flexibly. Several programming languages such NetCore, Trema and Frenetic have been proposed for configurations of OpenFlow controllers. NetCore can be used in Haskell, which is a functional programming language. There are several features of NetCore. NetCore is a declarative programming language and has a module structure. Programming in NetCore can be verified on Coq, which is a formal proof management system. This paper uses Coq to verify NetCore programmings for networks that consist of several OpenFlow Switches. Especially, this paper focus on verification of loop-free networks.

1. はじめに

柔軟なパケット転送を行いたいという要求により，OpenFlow[2] と呼ばれる，スイッチを制御するためのプロトコルが開発された．この OpenFlow に対応したスイッチは自分自身に制御部を持たず，与えられたメッセージのみに則って通信を行う．各スイッチにメッセージを送る機器はコントローラと呼ばれ，スイッチから送られるメッセージを読み取った際，プログラミングされた転送ルールをもとに，そのスイッチが行うべき各パケットに対する処理内容を送信する．

OpenFlow プロトコルを用いない従来のスイッチを導入して大規模なネットワークを構成する時，各スイッチに対して設定を行うにはコストがかかると考えられる．また，OpenFlow でないメーカー独自のプロトコルに対応したスイッチを導入したとしても，後で異なるベンダのスイッチを同様に制御して使用することは困難な作業となる．そのため，SDN(Software-Defined Network) と呼ばれる，ネットワーク構成をソフトウェアにより定義するアイデアが考案された．その実装技術の一つとして，2007 年に OpenFlow プロトコルが発表された．

従来の L2，L3 スイッチと同様，OpenFlow に対応したスイッチもパケットの転送は行う一方，その処理方法を生成しない．パケットの処理方法は OpenFlow プロトコルに基づいたメッセージが，コントローラと呼ばれる機器に送信され，コントローラが処理することにより生成される．スイッチは，コントローラから送られた命令に基づいてパ

†1 現在，埼玉大学工学部情報システム工学科
Presently with Departement of Information and Computer Sciences, Saitama University
a) hdate@fmx.ics.saitama-u.ac.jp
b) yoshiura@fmx.ics.saitama-u.ac.jp

ケット処理を行う．コントローラとスイッチの相互作用によりパケットが処理されるので，ネットワークの構成変更はコントローラへの設定で済ませることができる．

汎用的な計算機をコントローラとして動作させるために，OpenFlow コントローラの転送ルールをプログラミングすることを目的として，NetCore[3]，Trema[4]，Frenetic[5] など様々な言語やソフトウェアが開発されてきた．これらのうち，NetCore とは関数型言語 Haskell[6] でプログラミングできる言語であり，宣言型であること，モジュル構造を持っているなどの特徴がある．また，NetCore で記述したプログラムをコンパイル，実行することでコントローラを運用できる．宣言型であることから，パケットに対する具体的な処理内容を記述する必要がない．この特徴がプログラム検証には向いており，NetCore で記述されたプログラムの検証を，定理証明支援ソフト Coq[7] で行う研究が行われた．NetCore に対する研究として，そのプログラムの検証に関する研究が行われた [8]．この研究により，NetCore によりプログラミングされた言語が，プログラマの意図した通りにパケットを転送できるかを検証する事が可能となった．Coq 上での NetCore の検証は，記述したプログラムの仕様通りにパケットが到達できるか，または遮断できるかを判定することである．より具体的に記すと，NetCore プログラム pg が処理するパケットの前提条件を P，事後条件を Q とした時，Q を満たすようなパケットを，仕様 $\{P\}\,pg\,\{Q\}$ のプログラムが転送するための最弱前提条件 $wp\,(pg, Q)$ を P が満たしていることを証明することで行われた．この研究により，構築したプログラムの仕様通りにパケットが処理されることを強力に保証することが可能になった．ただし，実際に検証されたプログラムはスイッチが一つのトポロジーを対象として定義されており，複数スイッチが存在している場合での検証は行われていなかった．

そこで，本稿では，複数のスイッチが存在している場合のプログラムの検証を行う．ネットワークトポロジーにおいて，スイッチが 3 つ以上存在し，かつスイッチの接続で閉路が存在しているとき，適切でない転送ルールを導入するとパケットのループが発生する場合がある．同一パケットのループは，スイッチに不必要な負荷を与え，またネットワーク全体での転送速度の低下も発生しうる．そのため，そのようなトポロジーに対し，どのように適切なルールを敷設するかが重要となる．NetCore により，ループが発生しないよう設定された転送ルールを作成した時，Coq を用いてパケット転送が適切に行われることを証明することで，ループのないプログラムを作成できることを示すことを目指す．

以下，第 2 章で NetCore プログラムについて説明し，第 3 章では，Coq を用いた NetCore プログラムの検証方法を説明する．第 4 章では，実際に検証したプログラムと検証結果を示し，その結果と検証を行う過程への考察を行う．第 5 章で本稿をまとめる．

2. NetCore プログラム

ここでは，Coq における NetCore プログラムの作成方法を示す．NetCore は宣言型言語であるため，プログラマーはパケットに対して行いたい処理のみを記述すればよい．例えば，あるスイッチがポート 2 からパケットを受け取った時は，ポート 1 から転送し，それ以外からのパケットは遮断するような振る舞いを定義したいとき，まず以下のようなプログラムを導入できる．

```
Definition pg1 := WILD /=> FWD 1.
```

この式は，「どのパケットもポート 1 から転送する」というプログラムを表している．“Definition pg1 :=” の部分は，プログラムの名前を pg1 と定義している．“/=>” を挟んでいる項のうち左側は条件を指定でき，右側はその条件に合致したパケットに対する処理を指定できる．ここでは “WILD” は真，言い換えると全てのパケットが対象であることを表している．“FWD 1” は，「ポート 1 から転送する」という処理を表している．このプログラムだけではまだ仕様を満たせないので，別のプロブラムも用意する必要がある．

```
Definition pg2 := RESTRICT pg1 BY PORT=2.
```

この式は，「受信ポートが 2 だった時のみ pg1 を行う」という処理を表している．“RESTRICT x BY y” は，パケットが y を満たしている時だけプログラム x を動作させるよう指定する事ができる．“PORT=1” は受信ポートを表し，ここではポート 1 となっている．プログラム pg2 を転送ルールとして適応させることで，上記の振る舞いをさせることができる．

Coq において，指定できるパケットの条件や処理は他にも用意されており，静的 NAT やファイヤーウォールのプログラムも定義できる．

3. 検証方法

Coq において定義したプログラムが仕様通りに動作することを，どのように証明できるかを示す．

まず，想定している NetCore プログラムの仕様の証明を記述する際は，ホーア論理の形式で行う [1]．

$$\{P\}\,pg\,\{Q\}$$

この式において，P はスイッチに届いたパケットの満たす前提条件，pg は NetCore プログラム，Q は pg により処理されたパケットが満たす事後条件を表す．プログラムの仕様を検証するとき，この形式で記述された仕様が成り立

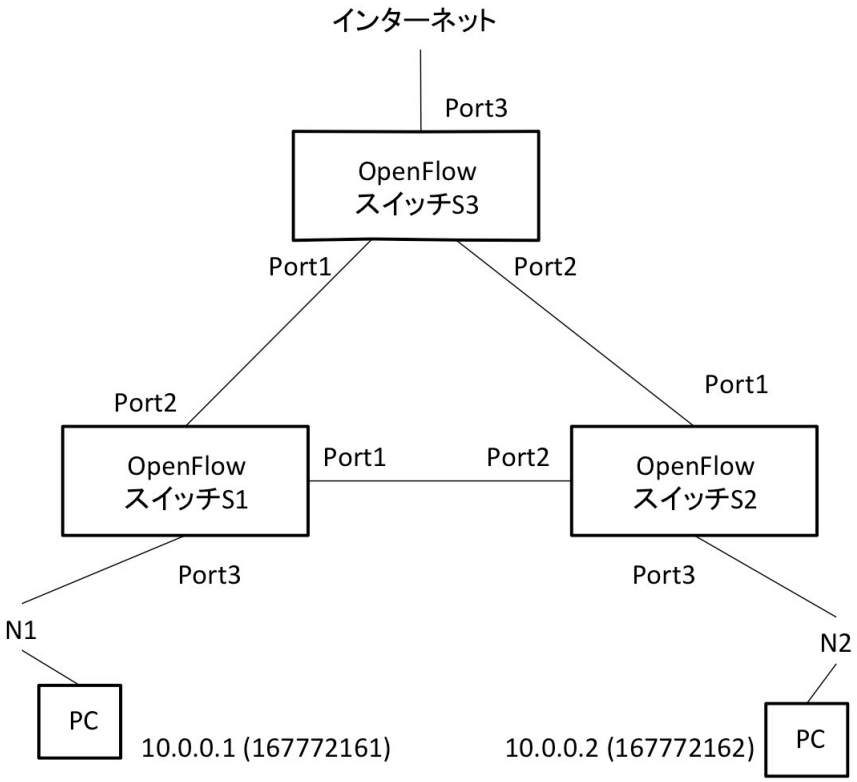

図 1 ネットワーク図

```
Load loadpath.
Require Import ZArith.
Require Import NetCoreWP.

Local Open Scope Z_scope.

(*s1_p1*)
Definition pg101 := WILD /=> FWD 1.
Definition pg102 := RESTRICT pg101 BY ((PORT=2 'OR' PORT=3) 'AND' NOT PORT=1 'AND' NWDST=167772162).

(*s1_p2*)
Definition pg103 := WILD /=> FWD 2.
Definition pg104 := RESTRICT pg103 BY (PORT=3 'AND' NOT NWDST=167772162).

(*s1_p3*)
Definition pg105 := WILD /=> FWD 3.
Definition pg106 := RESTRICT pg105 BY ((PORT=1 'OR' PORT=2) 'AND' NOT PORT=3 'AND' NWDST=167772161).

(*s1_behave*)
Definition pg100 := RESTRICT (pg102 'PAR' pg104 'PAR' pg106) BY SWITCH=1.

Goal |-[SWITCH=1 'AND' PORT=2 'AND' NWSRC=167772164 'AND' NWDST=167772162]pg100[PORT=1].
Proof. Time checker. Qed.


(*s2_p1*)
Definition pg201 := WILD /=> FWD 1.
Definition pg202 := RESTRICT pg201 BY ((PORT=2 'OR' PORT=3) 'AND' NOT PORT=1).

(*s2_behave*)
Definition pg200 := RESTRICT pg202 BY SWITCH=2.

Goal |-[SWITCH=2 'AND' PORT=2 'AND' NWSRC=167772164 'AND' NWDST=167772162]pg200[PORT=1].
Proof. Time checker. Qed.

Goal |-[SWITCH=2 'AND' PORT=1]pg200[NOT WILD].
Proof. Time checker. Qed.

(*s3_p1*)
Definition pg301 := WILD /=> FWD 1.
Definition pg302 := RESTRICT pg301 BY ((PORT=2 'OR' PORT=3) 'AND' NOT PORT=1 'AND' (NWDST=167772161 'OR' NWDST=167772162)).

(*s3_p3*)
Definition pg303 := WILD /=> FWD 3.
Definition pg304 := RESTRICT pg303 BY PORT=1.

(*s3_behave*)
Definition pg300 := RESTRICT (pg302 'PAR' pg304) BY SWITCH=3.

Goal |-[SWITCH=3 'AND'PORT=2 'AND' NWSRC=167772164 'AND' NWDST=167772162]pg300[PORT=1].
Proof. Time checker. Qed.
```

図 2 ループを起こしているプログラムとその検証結果

```
Load loadpath.
Require Import ZArith.
Require Import NetCoreWP.

Local Open Scope Z_scope.

(*s1_p1*)
Definition pg101 := WILD /=> FWD 1.
Definition pg102 := RESTRICT pg101 BY (PORT=3 'AND' NWDST=167772162 'AND' IS_IP).

(*s1_p2*)
Definition pg103 := WILD /=> FWD 2.
Definition pg104 := RESTRICT pg103 BY (PORT=3 'AND' (NOT NWDST=167772162 'OR' NOT IS_IP)).

(*s1_p3*)
Definition pg105 := PORT=1 /=> FWD 3.
Definition pg106 := PORT=2 /=> FWD 3.
Definition pg107 := RESTRICT (pg105 'PAR' pg106) BY ((NWDST=167772161 'OR' NOT IS_IP) 'AND' NOT PORT=3).

(*s1_behave*)
Definition pg100 := RESTRICT (pg102 'PAR' pg104 'PAR' pg107) BY SWITCH=1.

Lemma tst1: |-[SWITCH=1 'AND' PORT=3 'AND' NWDST=167772162 'AND' IS_IP]pg100[PORT=1].
Proof. Time checker. Qed.

Lemma tst1': |-[SWITCH=1 'AND' PORT=3 'AND' NOT NWDST=167772162 'AND' NOT NWDST=167772161]pg100[PORT=2].
Proof. Time checker. Qed.

Lemma tst1'':|-[SWITCH=1 'AND' NOT PORT=3 'AND' (PORT=1 'OR' PORT=2) 'AND' NWDST=167772161]pg100[PORT=3].
Proof. Time checker. Qed.

Lemma ether1: |-[SWITCH=1 'AND' PORT=3 'AND' NOT IS_IP]pg100[PORT=2].
Proof. Time checker. Qed.

Lemma ether1': |-[SWITCH=1 'AND' PORT=2 'AND' NOT IS_IP]pg100[PORT=3].
Proof. Time checker. Qed.

Lemma ether1'': |-[SWITCH=1 'AND' PORT=1 'AND' NOT IS_IP]pg100[PORT=3].
Proof. Time checker. Qed.

(*s2_p1*)
Definition pg201 := WILD /=> FWD 1.
Definition pg202 := RESTRICT pg201 BY (PORT=3 'AND' (NOT NWDST=167772161 'OR' NOT IS_IP)).

(*s2_p2*)
Definition pg203 := WILD /=> FWD 2.
Definition pg204 := RESTRICT pg203 BY (PORT=3 'AND' NWDST=167772161 'AND' IS_IP).

(*s2_p3*)
Definition pg205 := PORT=1 /=> FWD 3.
Definition pg206 := PORT=2 /=> FWD 3.
Definition pg207 := RESTRICT (pg205 'PAR' pg206) BY ((NWDST=167772162 'OR' NOT IS_IP) 'AND' NOT PORT=3).

(*s2_behave*)
Definition pg200:= RESTRICT (pg202 'PAR' pg204 'PAR' pg207) BY SWITCH=2.
```

図 3 ループを起こさないプログラムとその検証結果 その 1

```
Lemma tst2: |-[SWITCH=2 'AND' PORT=3 'AND' NOT NWDST=167772161 'AND' NOT NWDST=167772162]pg200[PORT=1].
Proof. Time checker. Qed.

Lemma tst2': |-[SWITCH=2 'AND' PORT=3 'AND' NWDST=167772161 'AND' IS_IP]pg200[PORT=2].
Proof. Time checker. Qed.

Lemma tst2'': |-[SWITCH=2 'AND' NOT PORT=3 'AND' (PORT=1 'OR' PORT=2) 'AND' NWDST=167772162]pg200[PORT=3].
Proof. Time checker. Qed.

Lemma ether2: |-[SWITCH=2 'AND' PORT=3 'AND' NOT IS_IP]pg200[PORT=1].
Proof. Time checker. Qed.

Lemma ether2': |-[SWITCH=2 'AND' PORT=1 'AND' NOT IS_IP]pg200[PORT=3].
Proof. Time checker. Qed.

Lemma ether2'': |-[SWITCH=2 'AND' PORT=2 'AND' NOT IS_IP]pg200[PORT=3].
Proof. Time checker. Qed.

(*s3_p1*)
Definition pg301 := WILD /=> FWD 1.
Definition pg302 := RESTRICT pg301 BY (PORT=3 'AND' NWDST=167772161).

(*s3_p2*)
Definition pg303 := WILD /=> FWD 2.
Definition pg304 := RESTRICT pg303 BY (PORT=3 'AND' NWDST=167772162).

(*s3_p3*)
Definition pg305 := WILD /=> FWD 3.
Definition pg306 :=
  RESTRICT pg305 BY ((PORT=1 'OR' PORT=2)'AND' NOT PORT=3 'AND' NOT NWDST=167772161 'AND' NOT NWDST=167772162).

(*s3_behave*)
Definition pg300 := RESTRICT (pg302 'PAR' pg304 'PAR' pg306) BY SWITCH=3.

Lemma tst3: |-[SWITCH=3 'AND' PORT=3 'AND' NWDST=167772161]pg300[PORT=1].
Proof. Time checker. Qed.

Lemma tst3': |-[SWITCH=3 'AND' PORT=3 'AND' NWDST=167772162]pg300[PORT=2].
Proof. Time checker. Qed.

Lemma tst3'': |-[SWITCH=3 'AND' (PORT=1 'OR' PORT=2) 'AND' NOT NWDST=167772161 'AND' NOT NWDST=167772162]pg300[PORT=3].
Proof. Time checker. Qed.

Remark other: |-[SWITCH=3 'AND' PORT=3 'AND' NOT NWDST=167772161 'AND' NOT NWDST=167772162]pg300[NOT WILD].
Proof. Time checker. Qed.
```

図 4 ループを起こさないプログラムとその検証結果 その 2

つことを証明するために，pg が Q を満たすパケットを転送するような，スイッチに届いたパケットが満たす最弱条件 $wp(pg, Q)$ を導出し，P がこの条件を含んでいるかを確認することが行われる．ここで，以前定義したプログラムが仕様通り動作するかを検証する．

```
Lemma veryfication: |-[PORT=2] pg2 [PORT=1].
Proof. checker. Qed.
```

この式において，“Lemma veryfication” は，証明 veryfication を行うことを表している．“|-[PORT=2] pg2 [PORT=1]．” はホーア論理で記述された仕様であり，ここでは「pg2 により処理されてポート 1 から転送されるパケットは，受信ポートが 2 である」ことを表している．そのような仕様の検証は，式の 2 行目の “checker.” が担う．checker は検証用に定義された関数であり，$wp(pg, Q)$ を自動で導出し，P と照会する．そして，プログラムの仕様が仕様が正しければ “No more subgoals.” を返して証明が完了するが，そうでなければエラーを返して停止する．ここで示した検証は，前者の振る舞いが行われることが確認できる．

このように，自分で設定したプログラムが仕様通りであるか，“Lemma...” のように検証することで証明することができる．

本稿では，図 1 にあるネットワークにおいて，OpenFlow コントローラに対する設定が，設定を記述した人の意図どおりに稼働すること，具体的には，ループが起きないことを検証する．

図 1 において，次のような設定を行う．

- OpenFlow スイッチ S3 では，Port1 と Port2 から受信したパケットは Port3 から送信する．Port3 から受信したパケットは Port1 と Port2 へ送信する．
- OpenFlow スイッチ S1 では，Port1 から受信したパケットは Port3 から送信する．Port2 から受信したパケットは Port3 へ送信する．Port3 から受信したパケットは Port1 と Port2 へ送信する．
- OpenFlow スイッチ S2 では，Port1 から受信したパケットは Port3 から送信する．Port2 から受信したパケットは Port3 へ送信する．Port3 から受信したパケットは Port1 と Port2 へ送信する．

この設定では図 1 においてパケットのループを発生させることはない．次章では，この設定を NetCore で記述し，パケットのループがないことを Coq を用いて証明する．さらに，故意にループを発生させるような設定を NetCore により記述し，Coq によりループの検出を行う．

4. 検証

図 2 は，図 1 における NetCore の記述の 1 つである．この記述，つまり，この設定では，パケットが 3 つのスイッチの間をループしてしまう．を生成する NetCore の記述であり，その検証結果を示したものである．この検証結果では，10.0.0.2 宛のパケットは，3 つのスイッチの間をループすることが分かる．

一方，図 3 と図 4 は，図 1 における別の NetCore の記述である．この記述，つまり，この設定の場合，パケットがループすることがないことを，図 3 と図 4 は示している．

図 2〜図 4 にある検証においては，CPU が Corei5-6600K，メモリが 16GB，OS は Ubuntu14.04LTS により実行した．これらの検証は 5 秒程度で終了する．

これらの検証は，ループの有無自体を直接検証しているわけではない．具体的には，特定の通信を想定して，その通信に対する各スイッチの処理を検証し，各スイッチの検証結果をまとめることでループの有無を検証している．よって，必ずしもネットワーク全体に対する検証とはなっていない．故に，検証を行おうとする場合，ネットワーク全体で検証したい性質を検証するために，各スイッチで検証すべき性質を導き出す必要がある．この導出自体が難しい可能性もある．実際，今回の図 3 と図 4 における検証を行う前に，ネットワーク全体を対象として，ループが発生しないことを検証することを検討していたが，Coq を利用するための論理式，つまり，ループが発生しないことを表現する論理式を見つけ出すことができなかった．また，検証したい性質を表す論理式を見つけたとしても，その検証が現実的な時間で終えることができるかが問題になるが，検証したい性質を表す論理式を導き出す方法が必要となる．

5. おわりに

本稿では，複数の OpenFlow スイッチからなるネットワークにおける OpenFlow コントローラの設定を，NetCore で記述したときに，その記述の検証を行った．しかし，個々の OpenFlow スイッチの検証を行うことによる検証であるため，ネットワーク全体をまとめて 1 つの検証を行うことが望まれる．今後は，複数の OpenFlow スイッチの性質を 1 つの検証，つまり，1 つの証明により行うための，性質の記述方法を検討する必要がある．

参考文献

[1] Hoare, C., A., R.: An axiomatic basis for computer programming, Communications of the ACM Vol.12, No.19, pp.576–580 (1969)
[2] McKeown, N., Anderson, T., Balakrishnan, H., Parulker, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J.: OpenFlow: Enabling Innovation in Campus Networks, “SIGCOMM Comput. Commun. Rev.”, Vol.38, No.2, pp.69–74 (2008)
[3] Monsanto, C., Foster, N., Harrison, R., and Walker, D.: A compiler and run-time system for network programming languages, Proceedings of the 39th annual

ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp.217–230 (2012)
[4] Trema. 入手先 ⟨http://trema.github.com/trema/⟩
[5] Frenetic. 入手先 ⟨http://www.frenetic-lang.org⟩
[6] Haskell. 入手先 ⟨https://www.haskell.org⟩
[7] The Coq Proof Assistant. 入手先 ⟨https://coq.inria.fr⟩
[8] Stewart, G.: Computational Verification of Network Programs in Coq, Proceedings of the Third International Conference on Certified Programs and Proofs, Lecture Notes in Computer Scirnce, Vol.8307, pp.33–49 (2013)