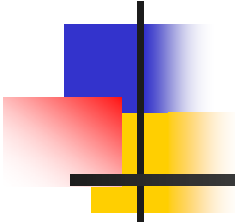


Title	定理証明によるファイアウォールサーバの検証
Author(s)	矢竹, 健朗
Citation	
Issue Date	2007-09-06
Type	Presentation
Text version	publisher
URL	http://hdl.handle.net/10119/8255
Rights	
Description	北陸先端科学技術大学院大学 21世紀COEシンポジウム 「検証進化可能電子社会」 = JAIST 21st Century COE Symposium “ Verifiable and Evolvable e-Society ” , 開催：2007年9月6日～7日, 開催場所：キャンパス・イ ノベーションセンター東京 国際会議室(1F), 2007年 9月6日(木), 「JAIST-COE/AIST-CVS シンポジウム ：形式検証技術—現状と安心電子社会への適用」発表 資料





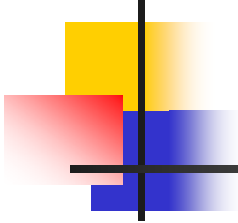
定理証明によるファイアウォールサーバの検証

2007.9.6

北陸先端科学技術大学院大学

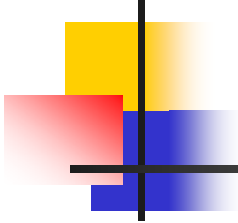
矢竹健朗

k-yatake@jaist.ac.jp



形式検証の必要性

- 情報システムの社会インフラ化。
 - システムの欠陥が社会に甚大な被害をもたらす。
→ システムの正しさを保証することが必要
- 問題点：システム検証はテスト主導。
 - 入力データは一般に無限。
 - 100%のカバレッジを実現することは不可能。
→ 厳密な数学理論に基づく網羅的な検証が必要。



定理証明によるシステム検証

■ 定理証明

- 最近、産業界でも注目されつつある網羅的検証技術の一つ。
- データに関する網羅的な検証が可能。
 - 「任意の n について $1+2+\dots+n = n*(n+1)/2$ 」
 - 「任意の入力データについてシステムが正しく動作する」
- 定理証明器：証明の計算機支援

手計算による証明



0からnまでの和を求める関数の定義
(Sum(0) = 0) ∧
(Sum(n+1) = n + 1 + Sum(n))

「Sum(n) = n*(n+1)/2」の証明
(証明)

nに関する帰納法を適用する。

(1) n=0のとき

$$(\text{左辺}) = \text{Sum}(0) = 0$$

$$(\text{右辺}) = 0 * (0+1) / 2 = 0$$

(2) nのとき成り立つと仮定

n+1のとき

$$\begin{aligned} (\text{左辺}) &= \text{Sum}(n+1) \\ &= n+1 + \text{Sum}(n) \\ &= n+1 + n*(n+1)/2 \end{aligned}$$

nの偶奇で場合分け

(2-1) n=2mのとき

$$\begin{aligned} (\text{左辺}) &= 2m+1 + 2m*(2m+1)/2 \\ &= 2m+1 + m*(2m+1) \\ &= (2m+1)(m+1) \end{aligned}$$

$$\begin{aligned} (\text{右辺}) &= (n+1)*(n+2)/2 \\ &= (2m+1)*(2m+2)/2 \\ &= (2m+1)(m+1) \end{aligned}$$

(2-2) n=2m+1のとき

$$\begin{aligned} (\text{左辺}) &= 2m+2 + (2m+1)*(2m+2)/2 \\ &= 2(m+1) + (2m+1)(m+1) \\ &= (m+1)(2m+3) \end{aligned}$$

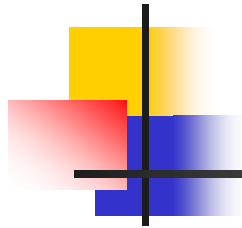
$$\begin{aligned} (\text{右辺}) &= (n+1)*(n+2)/2 \\ &= (2m+2)*(2m+3)/2 \\ &= (m+1)(2m+3) \quad (\text{証明終}) \end{aligned}$$

定理証明器による証明



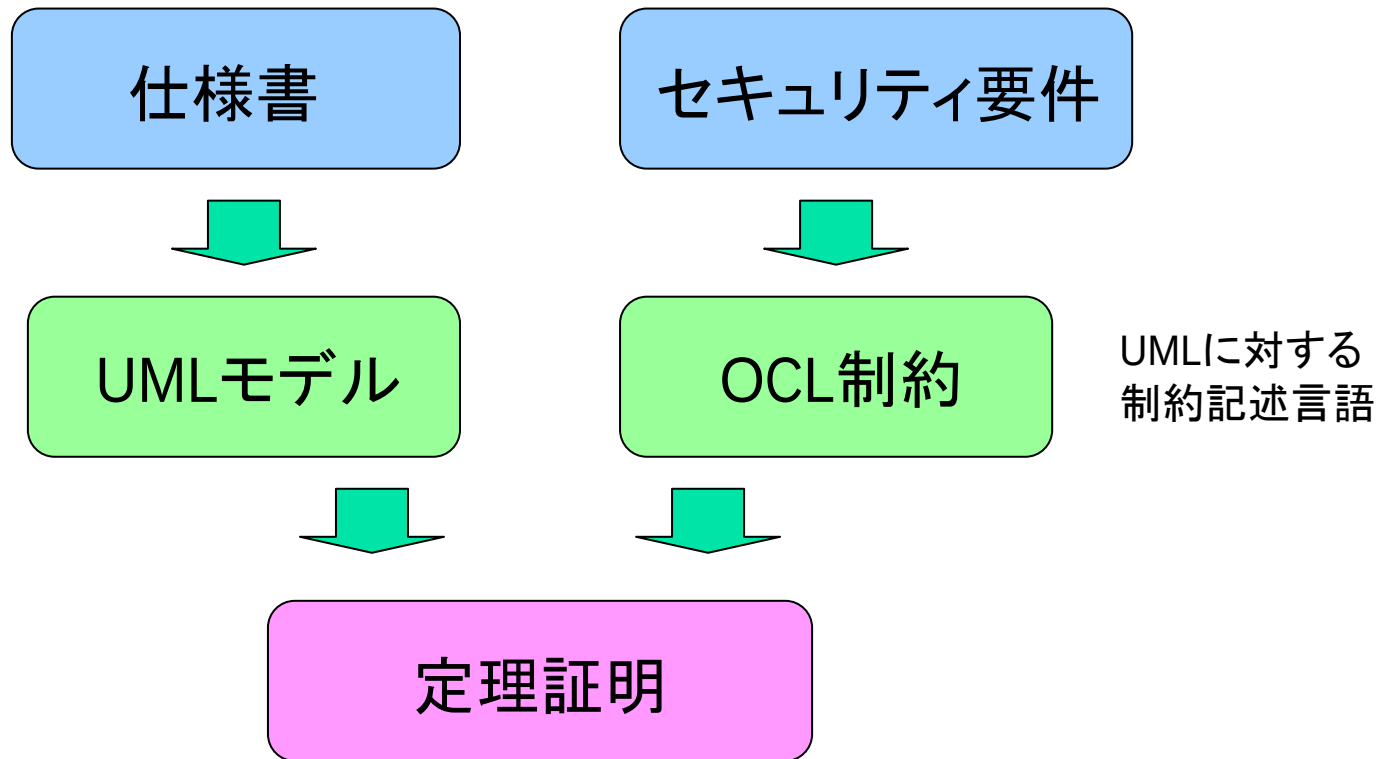
```
- Define `(Sum(0) = 0) ∧  
      (Sum(n+1) = n + 1 + Sum n)`;  
  (* Sumの定義 *)  
  
- g `!n. Sum n = n * (n + 1) / 2`;  
  (* ゴールの設定 *)  
> Initial goal:  
  !n. Sum n = n * (n + 1) / 2  
  
- e (Induct_on `n`);  
  (* nに関する帰納法の適用 *)  
> OK..  
2 subgoals:  
Sum (n+1) = (n+1) * (n+1+1) / 2  
-----  
Sum n = n * (n + 1) / 2  
  
Sum 0 = 0 * (0 + 1) / 2
```

```
- e (RW_TAC arith_ss [Sum]);  
  (* Sumの定義による書き換え *)  
> OK..  
Goal proved.  
|- Sum 0 = 0 * (0 + 1) / 2  
  
Remaining subgoals:  
Sum (n+1) = (n+1) * (n+1+1) / 2  
-----  
Sum n = n * (n + 1) / 2  
  
- e (RW_TAC arith_ss [Sum]);  
> OK..  
1 subgoal:  
n + 1 + n * (n + 1) / 2 = (n + 1) * (n + 2) / 2  
-----  
Sum n = n * (n + 1) / 2  
  
- ...
```

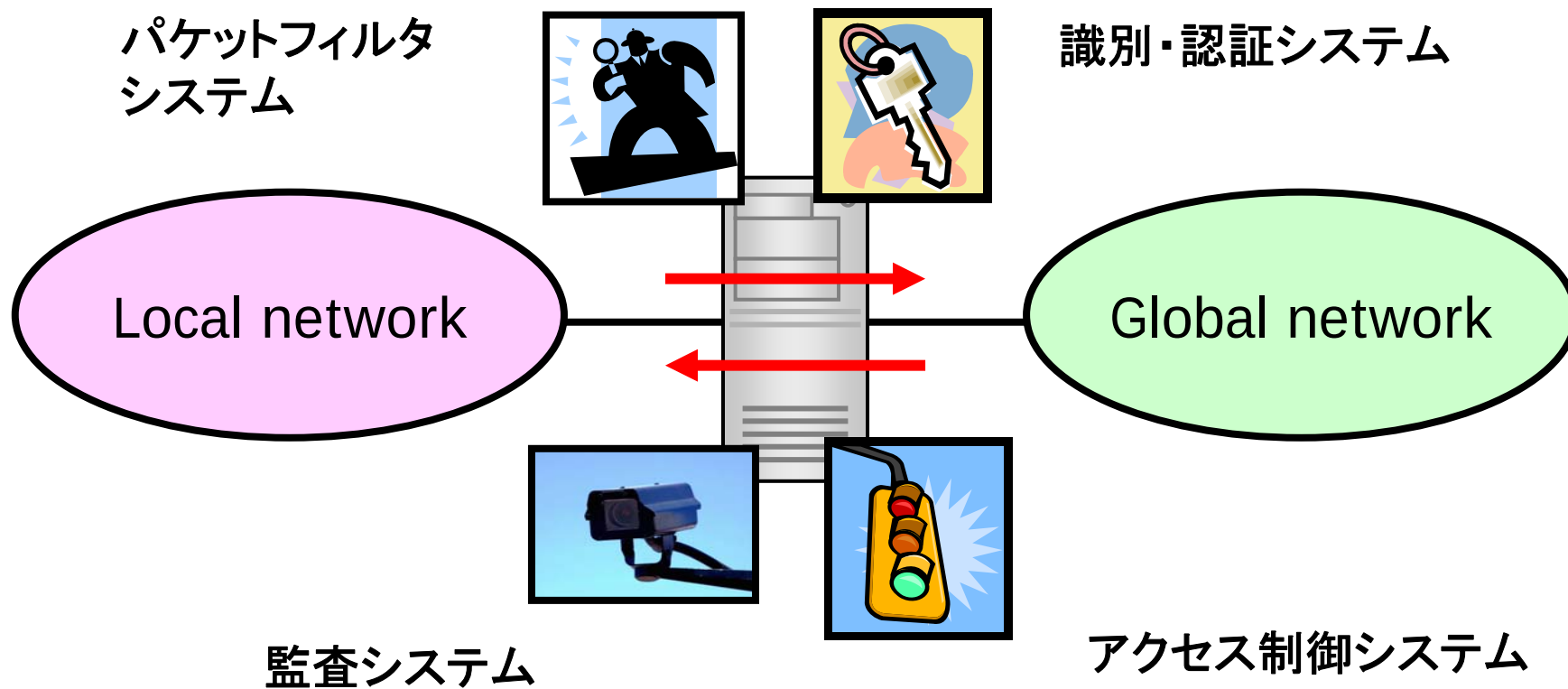


ファイアウォールの検証

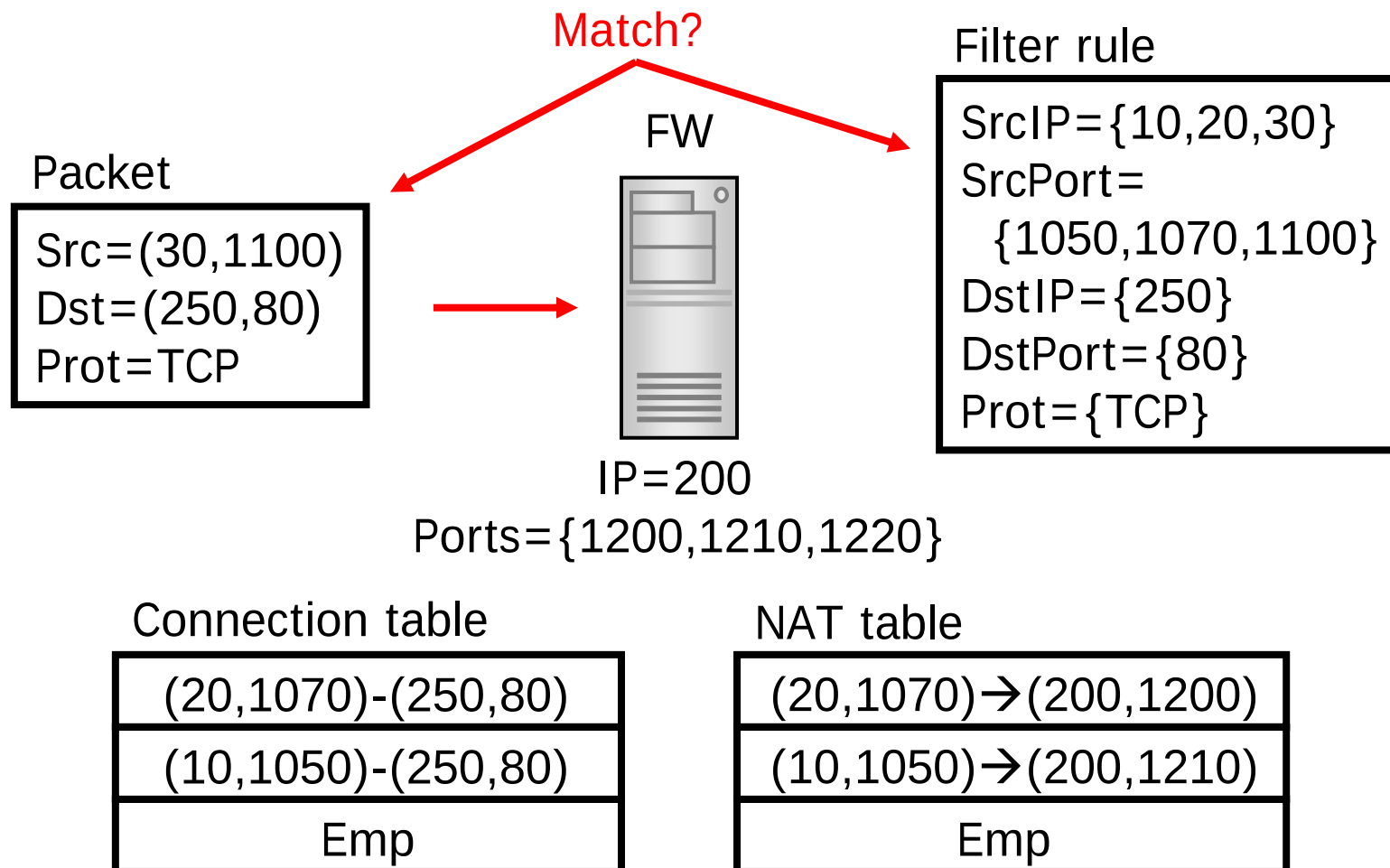
CISCO PIX Firewall
A4 5枚程度



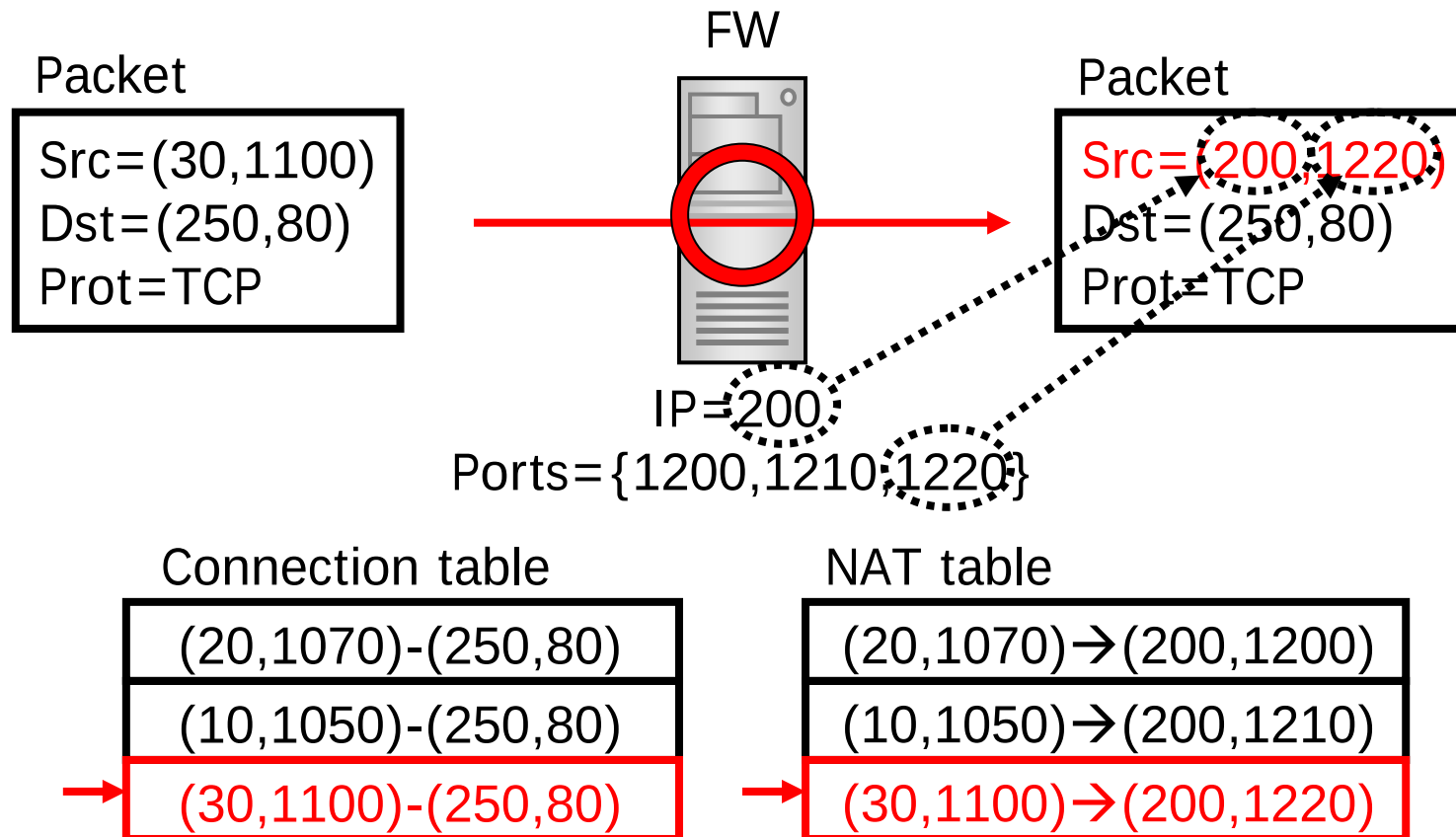
ファイアウォールの仕様

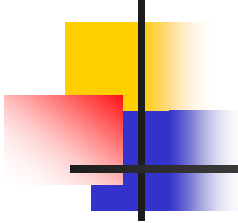


フィルタルール照合



NAT(ネットワークアドレス変換)

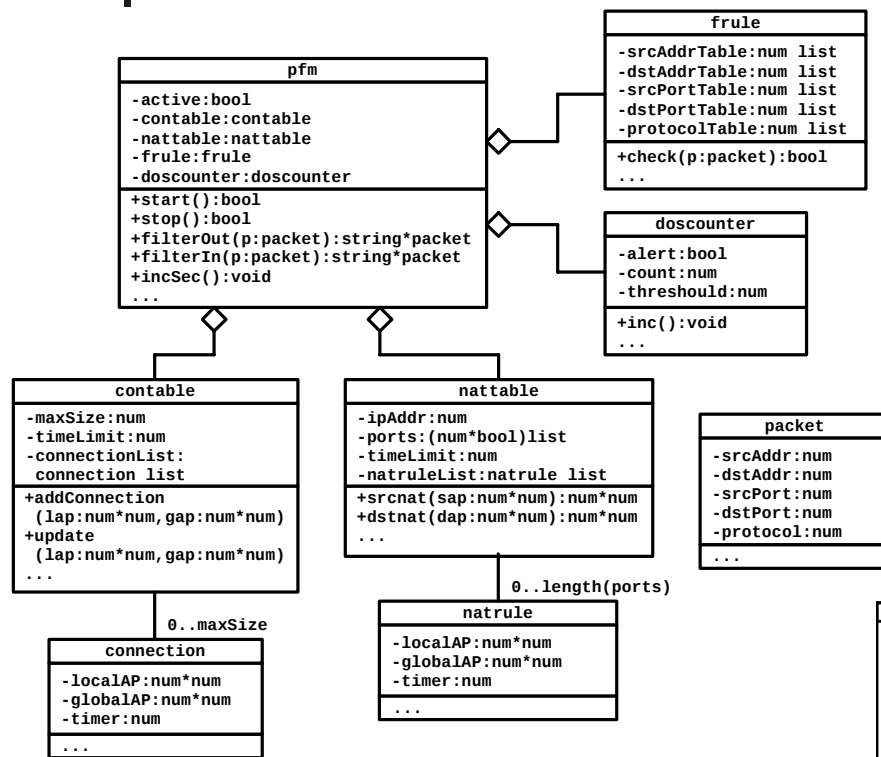




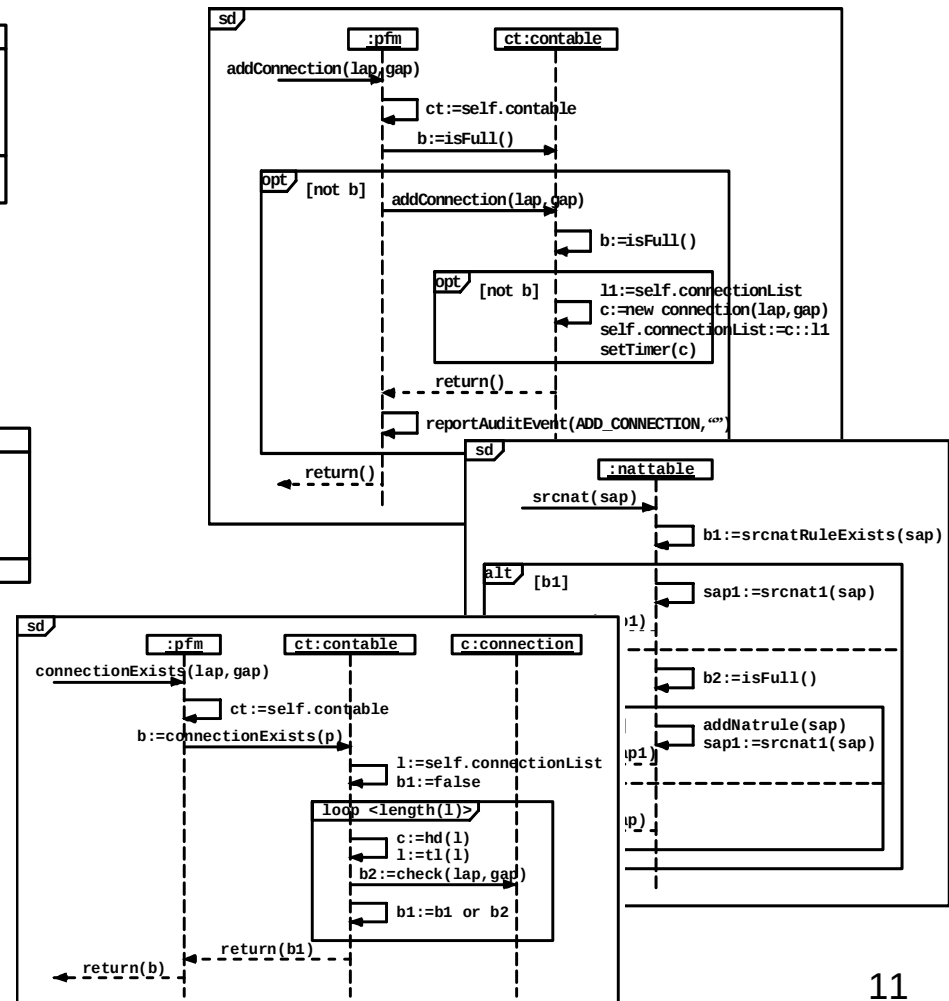
セキュリティ要件

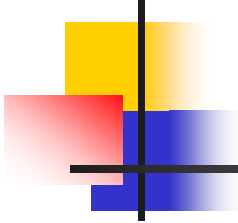
- フィルタルール照合の正当性
 - 新規のアウトバウンドパケットはフィルタルールを満たさない場合、必ず廃棄される。
- アドレス変換の正当性
 - パケットが外部ネットワークに送出されるとき、送信元IPアドレスが必ずFWの公開アドレスに書き換えられる。

UMLによるモデル化



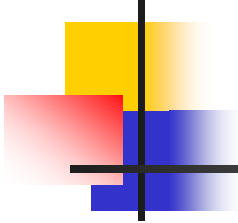
クラス図: 5枚
シーケンス図: 27枚





UMLによるモデル化

	クラス	属性	メソッド	API
パケットフィルタ	8	35	102	18
識別・認証	4	20	54	23
監査	4	13	37	10
アクセス制御	1	4	8	2
サブシステム統合部	2	10	66	50
ファイアウォール	19	82	267	50



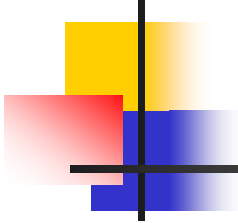
OCLセキュリティ要件

フィルタルール照合の正当性

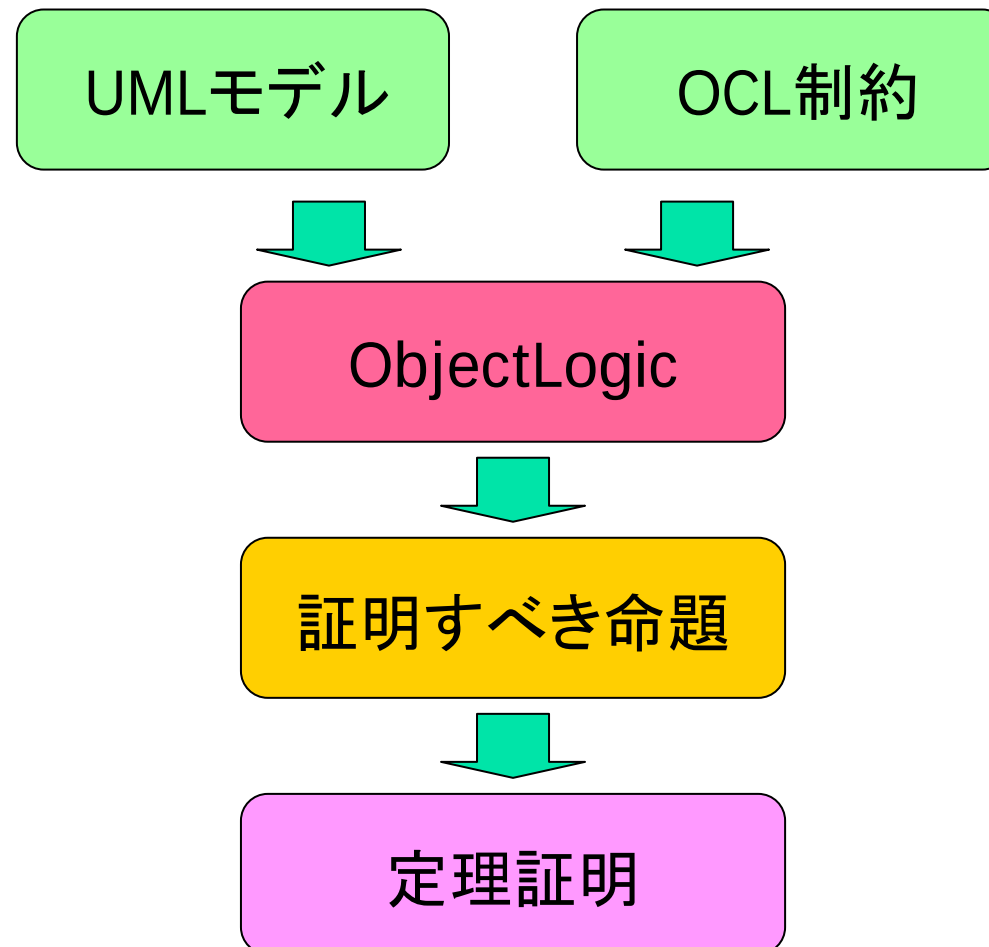
```
pfm::filterOut(p:packet):packet  
pre: not (isValidPacket(p))  
post: result=null
```

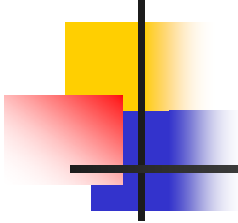
アドレス変換の正当性

```
pfm::filterOut(p:packet):packet  
post: not (result=null) implies  
      (result.srcAddr=pfm.getIpAddr())
```



UMLモデルの検証





定理証明

フィルタルール照合の正当性

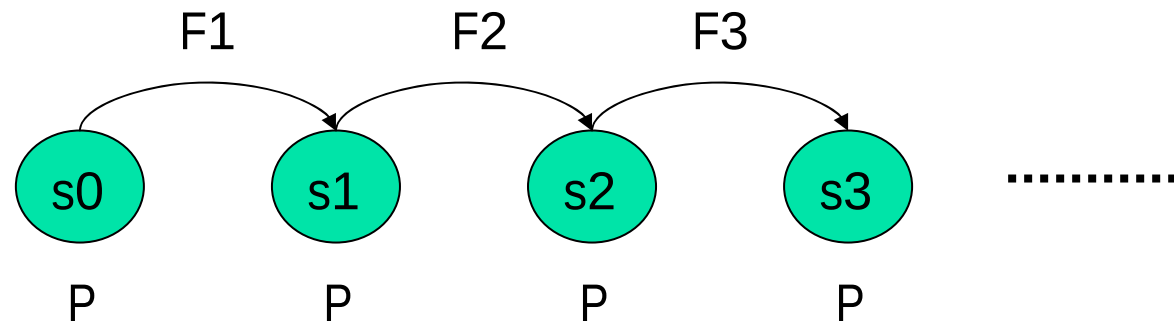
```
| - ∀ pfm p s.  
  let (p',s') = pfm_filterOut pfm p s in  
    packet_ex p s ⇒  
    ¬(pfm_isValidPacket pfm p s) ⇒ (p'=packet_null)
```

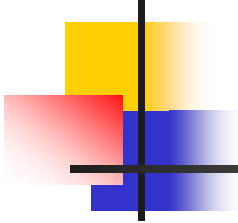
アドレス変換の正当性

```
| - ∀ pfm p s.  
  let (p',s') = pfm_filterOut pfm p s in  
    Inv pfm s ∧ ¬(p'=packet_null) ⇒  
    (packet_get_srcIpAddr p' s' = pfm_getIpAddr pfm s)
```

定理証明の有効性

- 網羅性。
 - データ、パス
- 不変表明の発見によるシステム内部制約の発見。
 - 不変表明: システムの実行中、常に成立している性質。
 - 定理の前提条件として必要になることが多い。
 - テストでは発見しにくい。





不変表明の例

- 接続表とアドレス変換表の一貫性。
 - 接続が存在するならば、その内部アドレスに対応するアドレス変換規則が必ず存在する。

接続表

(20,1070)-(250,80), 20
(10,1050)-(250,80), 30
(30,1100)-(250,80), 35

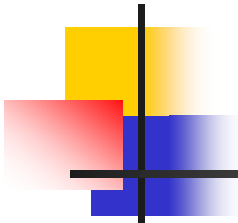
.....→

.....→

.....→

アドレス変換表

(20,1070)→(200,1200), 30
(10,1050)→(200,1210), 40
(30,1100)→(200,1220), 45



内部制約の発見

接続

(20,1070)-(250,80), 40



(20,1070)-(250,80), 20



(20,1070)-(250,80), 0

20 sec

20 sec

10 sec

アドレス変換規則

(20,1070)→(200,1200), 50



(20,1070)→(200,1200), 30

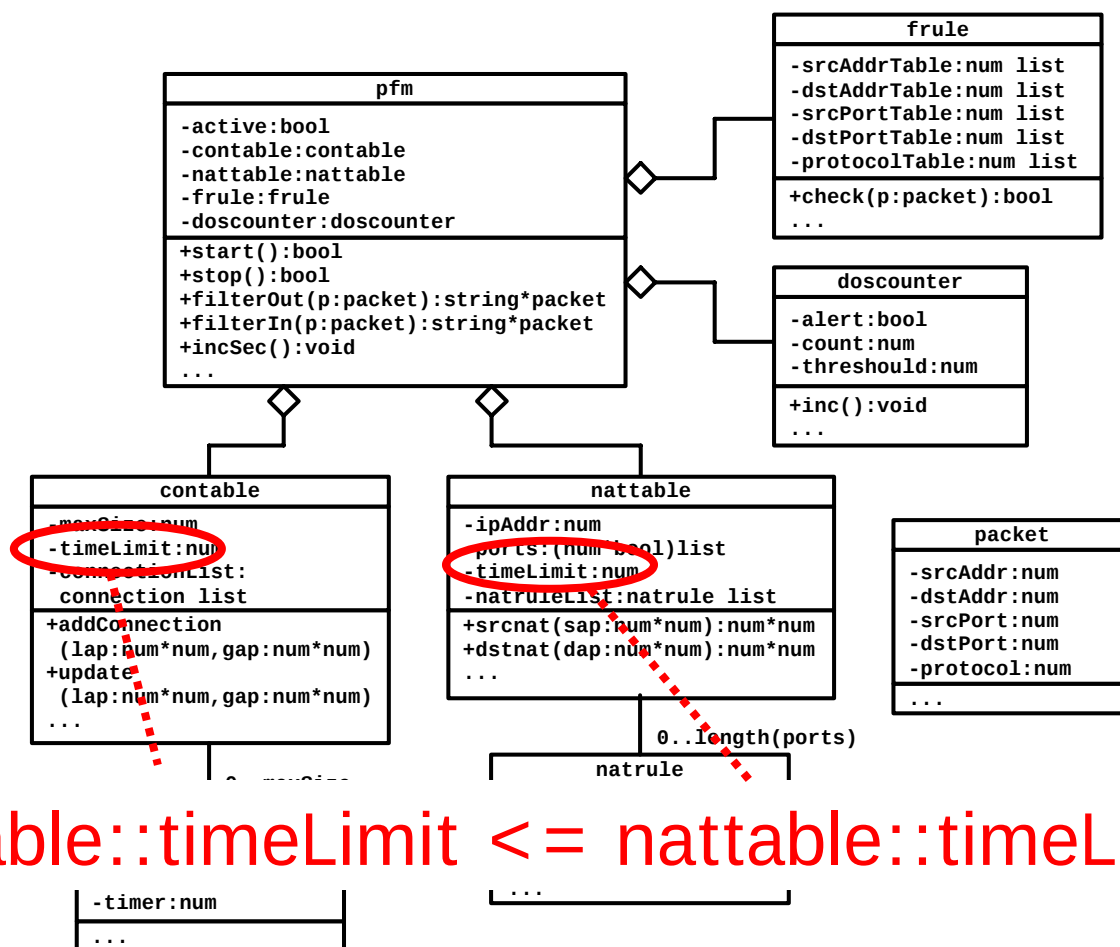


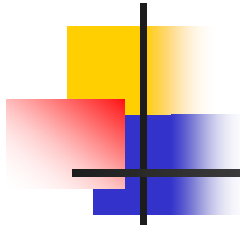
(20,1070)→(200,1200), 10



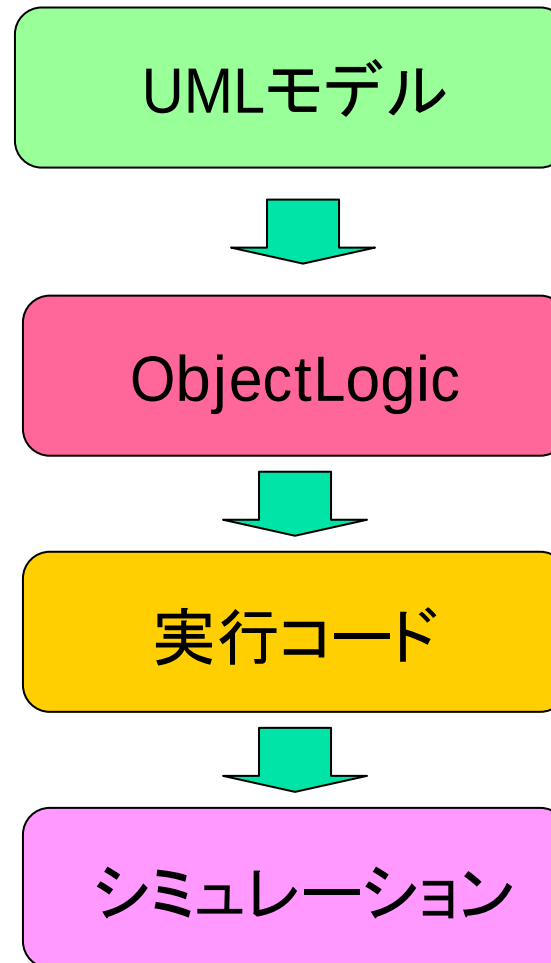
(20,1070)→(200,1200), 0

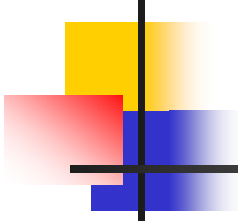
内部制約の発見





シミュレーション



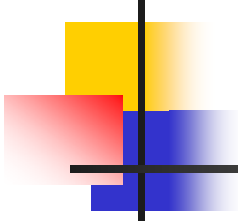


シミュレーションの有効性

- 簡単な誤りの発見。
 - 明らかに許可されるべきパケットが廃棄。
→ if文のthenとelseの処理が逆。
 - 新規パケットを送信したのに接続表が変化していない。
→ メソッド呼び忘れ。
- 高コストの定理証明の前に予め簡単な誤りを除去しておく。
→ 検証プロセス全体としてコストを削減。

シミュレーションの様子

- val (pfm,s) = new_pfm store_emp;		パケットフィルタの生成
...		
- val (_,s) = pfm_setIpAddr pfm 200 s;		
- val (_,s) = pfm_setPorts pfm [1200,1210,1220] s;	}	コンフィギュレーションの設定
...		
- val (_,s) = pfm_setFilterRules pfm SA [10,20,30] s;		
- val (_,s) = pfm_setFilterRules pfm SP [1050,1070,1100] s;	}	フィルタルールの設定
...		
- val (b,s) = pfm_start pfm s;		パケットフィルタの起動
...		
- val (p,s) = new_packet 20 1070 250 80 TCP s;		パケットの生成
...		
- val (msg,p,s) = pfm_filterOut pfm p s;		アウトバウンドフィルタの適用(許可)
> msg = "pass: new connection" : string		
...		
- pfm_getConnections pfm s;		
> val it = [((20,1070),(250,80),300)] : ((int*int)*(int*int)*int) list		
- pfm_getNatrules pfm s;		
> val it = [((20,1070),(200,1200),500)] : ((int*int)*(int*int)*int) list	}	パケットフィルタ、 パケットの状態の表示
- packet_getInfo p s;		
> val it = ((20,1200),(250,80),0) : (int*int)*(int*int)*int		
- val (p,s) = new_packet 20 1070 250 53 TCP s;		パケットの生成
...		
- val (msg,p,s) = pfm_filterOut pfm p s;		アウトバウンドフィルタの適用(廃棄)
> msg = "drop: rule not match" : string		
...		



まとめ

- 定理証明による実用的なファイアウォールサーバの検証。
 - ファイアウォールの仕様書、セキュリティ要件を、UMLモデルとOCL制約に直し、それが正しいことを定理証明により検証した。
- 定理証明の有効性
 - データ、パスに関する網羅性。
 - 不変表明の発見による内部制約の発見。