法令対象ドメインの形式仕様と検証

北陸先端科学技術大学院大学
情報科学研究科

二木厚吉　緒方和博　有本泰仁

お話しすること

1. 法令対象ドメインの形式記述の意味
2. ドメインの形式記述法
3. ドメインの形式記述と検証の例
4. まとめと今後の課題

法令対象ドメインの形式記述の意味

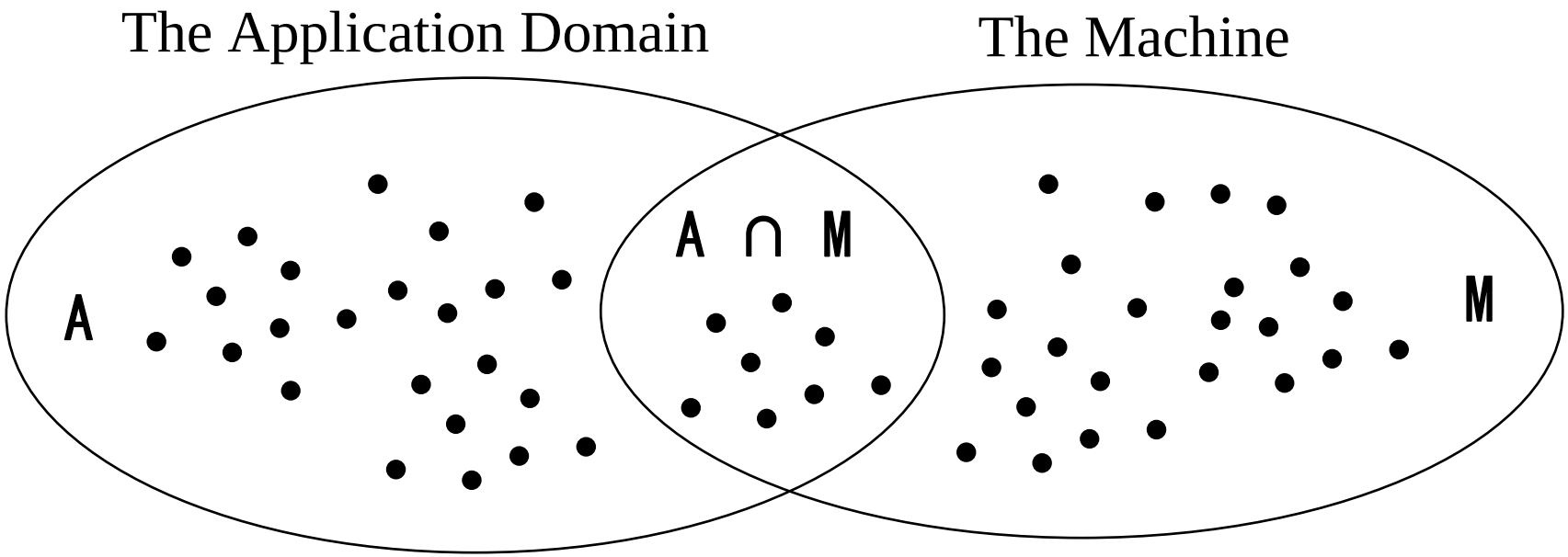
法令対象ドメイン記述の必要性

法令の系統的な作成・解析・検証・運用・保守のためには、法令文に記述された情報のみでなく，その法令が運用され意味を持つ領域（ドメイン）にたいする情報が必要

ソフトウェアシステムにおいても，そのシステムが運用され意味のある機能を果たすべきドメインを特徴付けるドメイン記述の重要性が認識されている

ドメインと機械　〈社会と運用〉

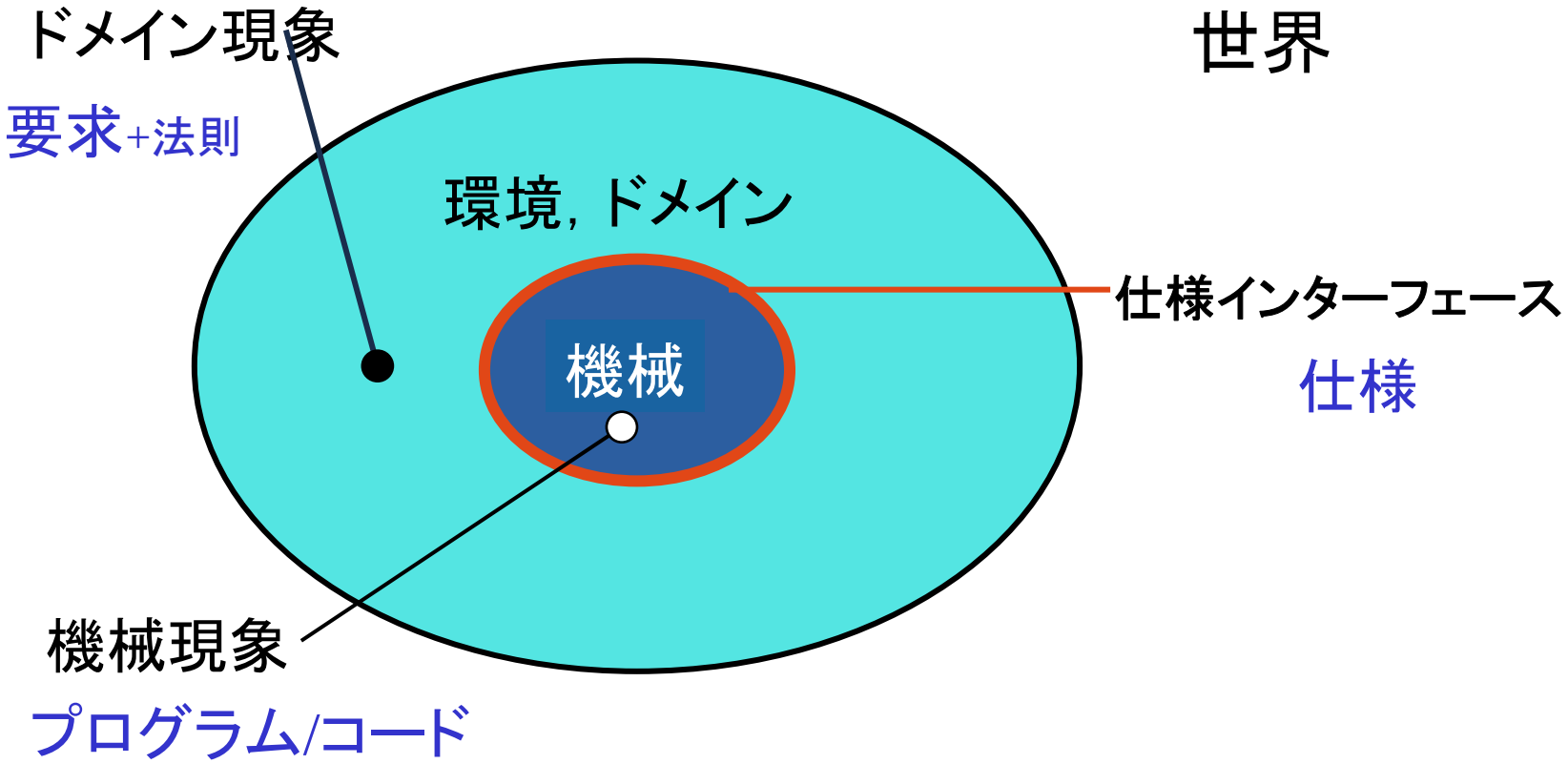
by Michael Jackson: Software Requirements & Specifications



<Society/Environment>　　<Law Execution/Practice>

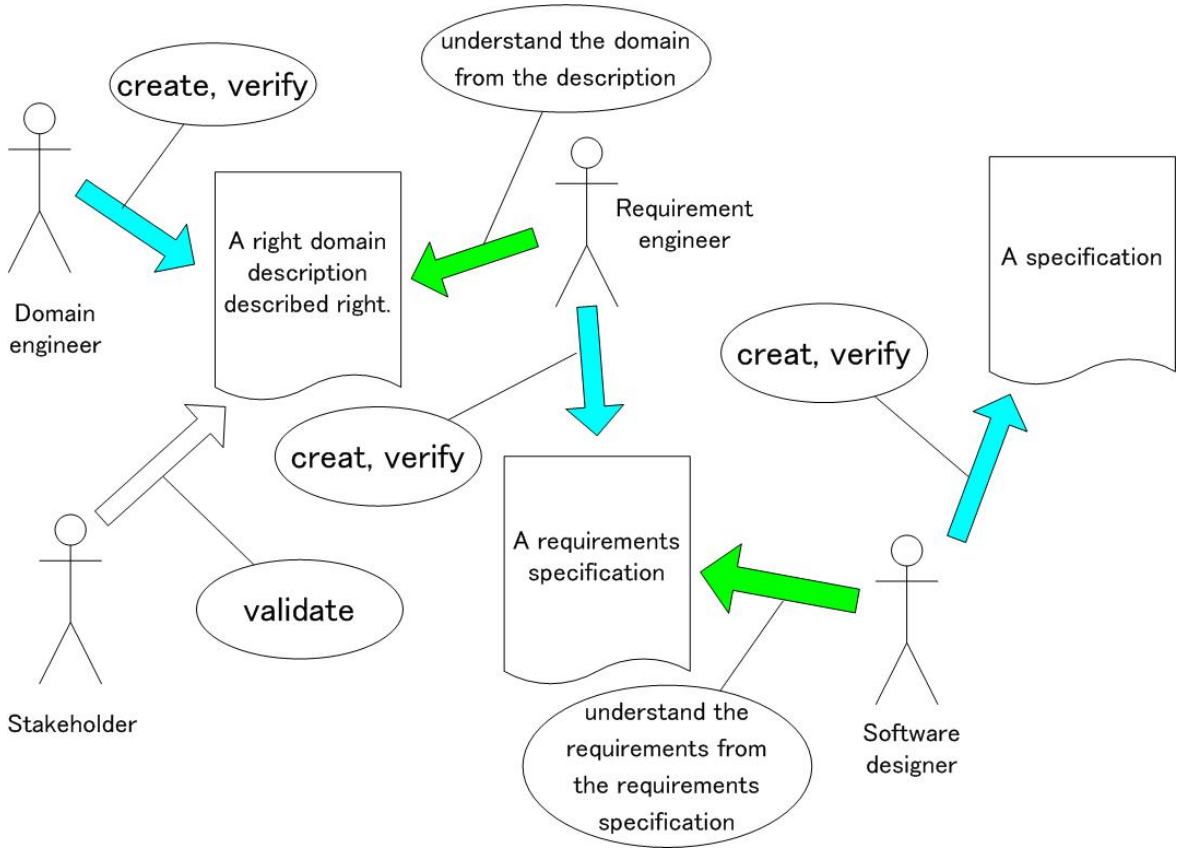
ドメイン，仕様，機械

by Michael Jackson: Software Requirements & Specifications



トリプティックドグマ(三面図教義)

トリプティックドグマ:ソフトウェアをデザイン，コード化する前に，要求をきちんと理解する必要があり，要求を適切に表現する前に，ドメイン(問題領域)をきちんと理解する必要がある．(Dines Bjorner)



法令対象ドメインにおける形式記述の意義

ドメインである意図を持って定められた法令，規則，
方針（ポリシー）の整合性や正当性などの解析・検証

客観的で厳密な解析や推論が可能であるドメイン
の形式記述(formal description)

形式記述の意味または最大の効用

- 問題領域中の概念の明確化と，用語の定義.
  - 用語：物の集まりとその上の操作の名前
    - Michael Jacksonの指示(designations)
    - 形式仕様での指標(signature)

何について話しているのか分からないのに，きっちりやろうなんて無意味だ．概念がはっきりしていなかったり，適用すべき対象の問題がはっきりしていないときに，正確な方法を使っても仕方ない．(John von Neumann)

形式化 == 厳密化
formal　　rigorous

ドメインの形式記述法

Dines Bjornerのドメイン記述法＋CafeOBJの形式記述法

Dines Bjornerのソフトウェア工学の教科書

Bjornerの教科書の概要 (2006)

www2.imm.dtu.dk/~db/Software_Engineering/

Contents of Three Volumes

[1] **Abstraction and Modelling**
- xxxix + 711 Pages

[2] **Specification of Systems and Languages**
- xxiv + 779 Pages

[3] **Domains, Requirements, and Software Design**
- xxx + 766 Pages

Prof. Dines Bjorner has stayed at JAIST from 2006.2 to 2007.1, and did the research on “Formal Methods and Domain Engineering”

ソフトウェア工学，要求工学，ドメイン工学

機械工学や化学工学などの他の工学が既に確立された問題領域を対象とするのに比べ，ソフトウェア工学の問題領域は特定されておらず，問題領域(domain)そのもののモデル化と分析・体系化がソフトウェア工学の大きな仕事になる．
➔ ドメイン工学，要求工学

形式と計算に基づくドメイン工学と要求工学がソフトウェア工学の重要な課題である

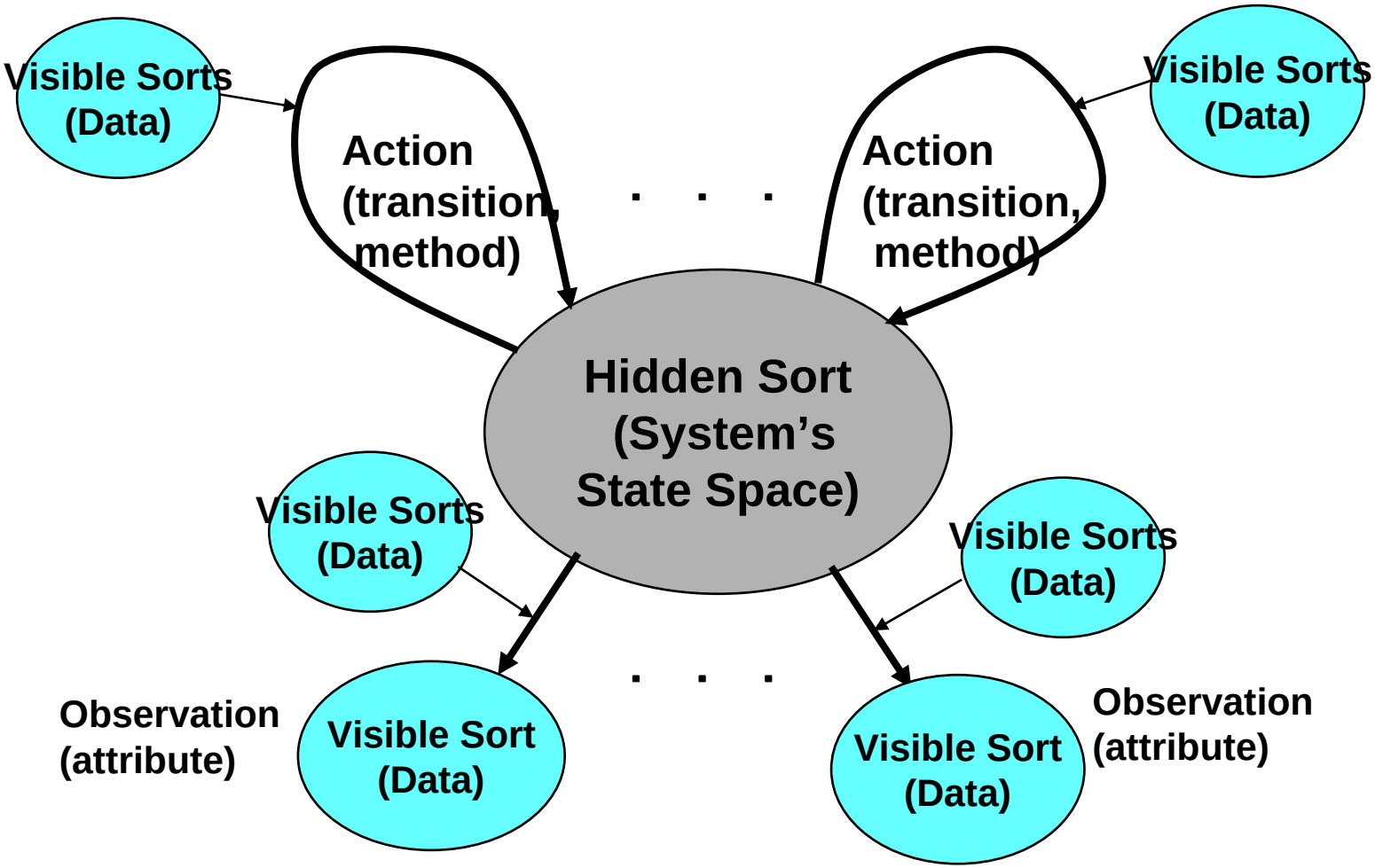
ドメインの記述：実体，関数，事象，振舞

ドメインの記述は「ドメインでの観測可能な現象の記述」であり，「観測可能な現象」とは，実体(entity)，関数(function)，事象(event)，そして振舞(behavior) のことであるとする。

- 実体： 実体とは対象とするドメインを構成する「もの」である．実体は型(typeまたはsort)により分類される．
- 関数： 関数はドメインに存在する種々の機能(function)を抽象化したものであり，実体を入力すると実体を出力する．関数を働かせて入力から出力を得ることを関数適用という．
- 事象： 事象はドメインに変化を引き起こす(つまりある時点の状態を入力として次の状態を出力とする) 関数適用である．
- 振舞： 振舞は事象の系列である．

CafeOBJにおける振舞(actionの系列)の記述法

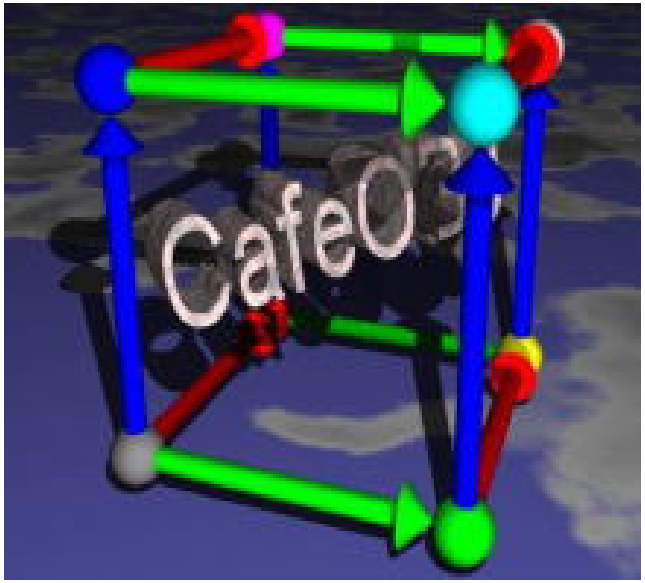
-- 振舞のOTS (Observational Transition System) によるモデル化 --



CafeOBJ

http://www.ldl.jaist.ac.jp/cafeobj

- 二木研究室を中心に国際チームにより研究開発された最先端の形式仕様言語システム(1995…)
- モデルを記述しその性質を解析・検証する実行可能モデル化言語システム
  - 実行可能形式仕様言語 システム (executable formal specification language system)
  - 対話型検証システム(interactive verification system)
  - 代数仕様言語システム(algebraic specification Language system)



CafeOBJ仕様はCafeOBJ言語システムの上で実行可能であり，その実行は仕様の意味を規定する等式推論に忠実である ➔ 検証の基盤

ドメインの形式記述法と検証の例

1. NSLPK認証プロトコル:プロトコルの検証が目的

2. 病院ドメイン: ドメインの形式記述(法の開発)が目的

NSLPK (Needham-Schroeder-Lowe Public-Key)
認証プロトコルの形式記述と検証

NSLPK Authentication Protocol

Msg1: p --> q $E_{k(q)}(n_p, p)$

Msg2: q --> p $E_{k(p)}(n_p, n_q, q)$

Msg3: p --> q $E_{k(q)}(N_q)$

- Each principal is given a pair of keys: public & private keys.

  `k(p)` is the public key given to `p`.

- N_p is a fresh nonce (random number) generated by `p`.
- The *(Nonce) Secrecy property*: Nobady except for `p` & `q` is not able to obtain N_p & N_q even if there are malicious principals.

Assumption

- The cryptosystem used is perfect; anyone cannot break it.
- There exists the general *intruder* a la Dolev-Yao.
- All principals except for the intruder exactly obey the protocol.
- What the intruder can do is

  1. to send and receive messages according to the protocol,
  2. to eavesdrop on every message that has been sent, and
  3. to fake messages based on wiretapped messages

  to cheat another principal into believing the intruder is yet another principle/intruder.

Formalizing Nonces

- A nonce made by a principal `p` to send it to a principal `q` is expressed as

  `n(p,q,r)`

  where `r` is a random number making the nonce unique.
- Operators `p1`, `p2` and `r` return the 1st, 2nd & 3rd args of `n(p,q,r)`.
- If the intruder obtains `n(p,q,r)` such that neither `p` nor `q` is the intruder somehow, the protocol does not satisfy the secrecy property.

Formalizing Ciphers

- $E_{k(q)}(n_p, p)$ is expressed as

  $\mathtt{enc1}(q, n_p, p)$

- $E_{k(p)}(n_p, n_q, q)$ is expressed as

  $\mathtt{enc2}(p, n_p, n_q, q)$

- $E_{k(q)}(n_q)$ is expressed as

  $\mathtt{enc3}(q, n_q)$

- Given a term denoting a cipher, operator `k` returns the principle ID for the public key, operator `n1` returns the 1st nonce, operator `n2` returns the 2nd nonce if any, and operator `p` returns the principal ID if any.

- Ciphers are treated as messages.

Formalizing Networks

- Networks are expressed as bags (multisets) of messages.
- Const `empty` & operator `_ _` are the data constructors of bags.
- Predicate `_¥in_` checks if a given message is in a given bag.
- The assumption that the intruder can eavesdrop on each message that has been sent is expressed as that the intruder can use the network used in the protocol as his/her storage.
- Since the intruder can replay any messages, each message that has been sent can keep staying in the network. So, each message that has been sent is never deleted from the network.

Modeling NSLPK as an OTS (1)

- 2 observations `rand`, `nw` & `nonces`:
  1. `rand(s)` returns a fresh random number that has never been generated up to a given state `s`.
  2. `nw(s)` returns the bag (the network) of messages sent up to a given state `s`.
  3. `nonces(s)` returns the bag of nonces generated for the sessions participated by the intruder up to a given state `s`.
- For an arbitrary initial state `init`,
  1. `rand(s)` returns `seed`, which is the first rand number.
  2. `nw(init)` returns `empty`.
  3. `nonces(init)` returns `empty`.

Modeling NSLPK as an OTS (2)

- 6 actions `send1`, `send2`, `send3`, `fake1`, `fake2` & `fake3`

`send1`, `send2` & `send3` model sending Msg1, Msg2 & Msg3 messages exactly obeying the protocol.

1. `send1(s,p,q)` denotes the next state after `p` has sent `enc1(q,`n_p`,p)` to `q` in `s`, where n_p is a fresh nonce.
2. `send2(s,p,q,n1)` denotes the next state after `p` has sent `enc2(q,n1,`n_p`,p)` to `q` in `s` if `enc1(p,n1,q)` is in the network.
3. `send3(s,p,q,n1,n2)` denotes the next state after `p` has sent `enc3(q,n2)` to `q` in `s` if `enc1(q,n1,p)` & `enc2(p,n1,n2,q)` are in the network.

Modeling NSLPK as an OTS (3)

`fake1`, `fake2` & `fake3` model the intruder's faking Msg1, Msg2 & Msg3 messages based on the gleaned nonces.

1. `fake1(s,p,q,n1)` denotes the next state after the intruder has faked `enc1(q,n1,p)` & sent it to `q` in `s` if `n1` is in `nonces(s)`.
2. `fake2(s,p,q,n1,n2)` denotes the next state after the intruder has faked `enc2(q,n1,n2,p)` & sent it to `q` in `s` if `n1` & `n2` are in `nonces(s)`.
3. `fake3(s,p,q,n1)` denotes the next state after the intruder has faked `enc3(q,n1)` & sent it to `q` in `s` if `n1` is in `nonces(s)`.

Specifying NSLPJ in CafeOBJ (1)

- The signature is as follows:

```
*[Sys]*
-- an arbitrary initial state
op init : -> Sys
-- observers
bop rand   : Sys -> Rand
bop nw     : Sys -> Network
bop nonces : Sys -> NonceBag
-- actions
bop send1 : Sys Prin Prin -> Sys
bop send2 : Sys Prin Prin Nonce -> Sys
bop send3 : Sys Prin Prin Nonce Nonce -> Sys
bop fake1 : Sys Prin Prin Nonce -> Sys
bop fake2 : Sys Prin Prin Nonce Nonce -> Sys
bop fake3 : Sys Prin Nonce -> Sys
```

Specifying NSLPK in CafeOBJ (2)

- The equations defining `send2` are as follows:

```
-- send2
  op c-send2 : Sys Prin Prin Nonce -> Bool
  eq c-send2(S,P1,P2,N1) = enc1(P1,N1,P2) ¥in nw(S) .
  --
  ceq rand(send2(S,P1,P2,N1)) = next(rand(S))
                                if c-send2(S,P1,P2,N1) .
  ceq nw(send2(S,P1,P2,N1))
      = (enc2(P2,N1,n(P1,P2,rand(S)),P1) nw(S))
        if c-send2(S,P1,P2,N1) .
  ceq nonces(send2(S,P1,P2,N1)) = (if P2 = intr then
           (N1 n(P1,P2,rand(S)) nonces(S)) else nonces(S) fi)
           if c-send2(S,P1,P2,N1) .
  ceq send2(S,P1,P2,N1) = S if not c-send2(S,P1,P2,N1) .
```

Specifying NSLPK in CafeOBJ (3)

- The equations defining `fake2` are as follows:

```
-- fake2
  op c-fake2 : Sys Prin Prin Nonce Nonce -> Bool
  eq c-fake2(S,P1,P2,N1,N2)
     = N1 ¥in nonces(S) and N2 ¥in nonces(S) .
  --
  eq  rand(fake2(S,P1,P2,N1,N2)) = rand(S) .
  ceq nw(fake2(S,P1,P2,N1,N2)) = (enc2(P1,N1,N2,P2) nw(S))
                                 if c-fake2(S,P1,P2,N1,N2) .
  eq  nonces(fake2(S,P1,P2,N1,N2)) = nonces(S) .
  ceq fake2(S,P1,P2,N1,N2) = S
                             if not c-fake2(S,P1,P2,N1,N2) .
```

Verifying the Secrecy Prop with CafeOBJ (1)

- The two modules (INV and ISTEP) are declared:

```
mod INV { pr(NSLPK)
  op n1 : -> Nonce
  op inv1 : Sys Nonce -> Bool
  var S : Sys  var N1 : Nonce
  eq inv1(S,N1) = ((N1 ¥in nonces(S)) implies
                   (p1(N1) = intr or p2(N1) = intr)) .
}

mod ISTEP { pr(INV)
  ops s s' : -> Sys
  op istep1 : Nonce -> Bool
  var N1 : Nonce
  eq istep1(N1) = inv1(s,N1) implies inv1(s',N1) .
}
```

Verifying the Secrecy Prop with CafeOBJ (2)

- Write the initial proof score for inv1(S,N1), which consists of 12 proof passages.
- CafeOBJ does not return true for the 2nd, 3rd & 5th proof passages.
- The 3rd proof passage is split into 7 sub-cases:

✔ 1. `not(intr = p2)`

✔ 2. `intr = p2, n1 = n(p1,p2,rand(s))`

✔ 3. `intr = p2, not(n1 = n(p1,p2,rand(s))), not(n1 = m)`

✔ 4. `intr = p2, not(n1 = n(p1,p2,rand(s))), n1 = m, p1(m) = p2`

✔ 5. `intr = p2, not(n1 = n(p1,p2,rand(s))), n1 = m, not(p1(m) = p2), p2(m) = p2`

✔ 6. `intr = p2, not(n1 = n(p1,p2,rand(s))), n1 = m, not(p1(m) = p2), not(p2(m) = p2), m ¥in nonces(s)`

? 7. `intr = p2, not(n1 = n(p1,p2,rand(s))), n1 = m, not(p1(m) = p2), not(p2(m) = p2), not(m ¥in nonces(s))`

Verifying the Secrecy Prop with CafeOBJ (3)

```
open ISTEP
-- arbitrary values
  ops p1 p2 : -> Prin .  op m : -> Nonce .
-- assumptions
  -- eq c-send2(s,p1,p2,m) = true .
  eq enc1(p1,m,p2) ¥in nw(s) = true .
  --
  eq intr = p2 .  eq (n1 = n(p1,p2,rand(s))) = false .
  eq n1 = m .  eq (p1(m) = p2) = false .
  eq (p2(m) = p2) = false .  eq m ¥in nonces(s) = false .
-- successor state
  eq s' = send2(s,p1,p2,m) .
-- check
  red istep1(n1) .
close
```

Verifying the Secrecy Prop with CafeOBJ (4)

- Conjecture the lemma:

```
eq inv2(S,N1,Q1)
   = ((enc1(Q1,N1,intr) ¥in nw(S))
      implies (p1(N1) = intr or p2(N1) = intr
                             or N1 ¥in nonces(S))) .
```

- The 2nd proof passage does not need any lemmas.
- The 5th proof passage needs the lemma:

```
eq inv3(S,N1,N2,Q1)
   = ((enc2(Q1,N1,N2,intr) ¥in nw(S))
      implies (p1(N2) = intr or p2(N2) = intr
                             or N2 ¥in nonces(S))) .
```

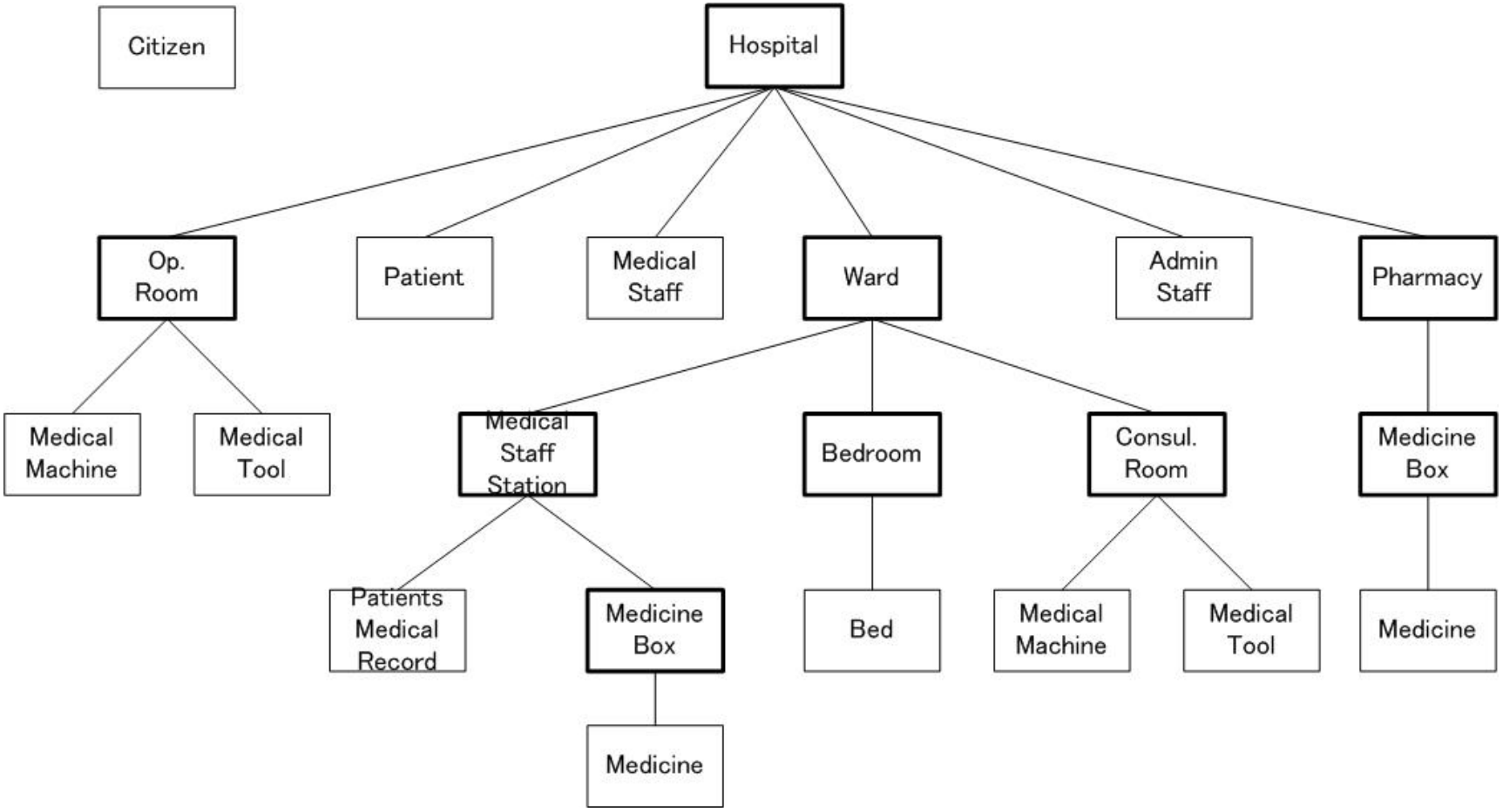
- The proofs of `inv2(S,N1,Q1)` & `inv3(S,N1,N2,Q1)` do not need any lemmas.

病院ドメインの形式記述

形式記述開発の手順

- 病院ドメインを例とした形式記述の開発事例
  - 自然言語による病院ドメインの記述(非形式記述)
    - Dines Bjornerの提唱している方法に則る
    - 形式化を意識して記述する
  - 病院ドメインのOTS (Observational Transition System)モデルの作成
  - OTSモデルを代数仕様言語CafeOBJで記述
  - 病院ドメインのCafeOBJ仕様に基づく簡単な形式検証

病院ドメインの実体


Citizen
Hospital
Op. Room
Patient
Medical Staff
Ward
Admin Staff
Pharmacy
Medical Machine
Medical Tool
Medical Staff Station
Bedroom
Consul. Room
Medicine Box
Patients Medical Record
Medicine Box
Bed
Medical Machine
Medical Tool
Medicine
Medicine

関数，アクション（OTSのactionとは異なる）

- 関数：
  - 実体や実体の属性に対して適用することによって
    - 実体の値を変化させるもの.
    - なんらかの計算結果を返すもの.
  - ある実体の値を観測するもの.
- 特に実体の値を変化させるものをアクションと呼ぶ.
- 病院ドメインでは次のようなアクションが考えられる：
  - admit : **Citizen × WardID ×Hospital → Hospital**
  - interview : **PatientID × Hospital → Hospital**
  - plan analysis : **PatientID × Hospital → Hospital**
  - analyse : **PatientID × Hospital → Hospital**
  - diagnose : **PatientID × Hospital → Hospital**
  - plan : **PatientID × Hospital → Hospital**
  - treat : **PatientID × Hospital → Hospital**
  - transfer :
    PatientID ×WardID × WardID × Hospital → Hospital
  - release : **PatientID × Hospital → Hospital**

事象，振舞

- 事象：
  - 起こることによって関数が適用される引き金，他の事象が起こる引き金となりうるもの，
  - または関数が適用されたことや他の事象が引き金となって起こりうるもの．
- 病院ドメイン上の事象は次のようなものが考えられる：
  - A citizen comes to a hospital
  - A patient gets cured.
  - A patient dies
  - A patient gets a new symptom

- 振舞：アクションと事象の列のこと．
- 病院ドメイン上での振舞いの１つとして次のようなものが考えられる：
  1. a citizen comes to a hospital,
  2. admit,
  3. interview,
  4. plan analysis,
  5. analyse,
  6. diagnose,
  7. plan,
  8. treat,
  9. analyse,
  10. a patient gets cured,
  11. release.

観測遷移システム
(OTS:Observational Transition System)

- OTS：システムの振舞をモデル化する状態遷移システム
- OTS S <O, I, T>の3組で表される
  - O ： 観測関数(observation)の有限集合
  - I ： 初期状態の集合
  - T ： 遷移演算(transition)の有限集合

- 病院ドメインの振舞をOTSでモデル化する

OTSモデル：観測

- 病院の状態を表すソートをHosとする.
- 観測関数は以下の値を観測する.
  - 病院の属性
    - 病院のID
    - 病院の住所
  - 病院の各下位実体の集合
    - 病棟の集合
    - 手術室の集合
    - 薬局の集合
    - 経営スタッフの集合
    - 患者の集合
    - 医療スタッフの集合
  - 市民の居場所
  - 患者の体調

```
-- attributes are th ID of the hospital and the location.
  bop h-id : Hos -> HID
  bop h-loc : Hos -> Loc
-- subentitis
  bop wset : Hos -> WSet
  bop orset : Hos -> ORSet
  bop phset : Hos -> PhSet
  bop asset : Hos -> ASSet
  bop paset : Hos -> PaSet
  bop msset : Hos -> MSSet
-- observer for a location of a citizen.
  bop cit-loc : CID Hos -> Loc
-- observer for a condition of a patient.
  bop p-cond : PID Hos -> Cond
```

OTSモデル：初期状態

- メレオロジー：ある合成実体が下位実体にどのように構成されているか表す概念.
- 初期状態の集合
  - 任意の初期状態を`init`とする.
  - 初期状態では病院のメレオロジーは満たされているものとする.

```
-- initial states
op init : -> Hos
eq (card(wset(init)) > 0) = true .
eq (card(orset(init)) > 0) = true .
eq (card(phset(init)) = 1) = true .
eq (card(asset(init)) > 0) = true .
eq (card(paset(init)) >= 0) = true .
eq (card(msset(init)) > 0) = true .
```

OTSモデル：遷移演算 = アクション or 事象

- 遷移演算の集合
  - アクションと事象

```
-- transitions

bop admit : Cit Wid Hos -> Hos

bop interview : PMRid Wid Hos -> Hos

bop planAnal : PMRid Wid Hos -> Hos

bop doAnal : PMRid Wid Hos -> Hos

...

bop cch : Cit Hos -> Hos -- a citizen comes to a hospital.

bop pgc : Pid Hos -> Hos -- a patient gets cured.

bop pgw : Pid Hos -> Hos -- a patient gets worse.

bop pd : Pid Hos -> Hos -- a patient dies.
```

OTSモデル：遷移演算の例admit

admit(C:Cit, W:WID, H:Hos)の事前条件

- 市民Cは病院の中にいる.
- IDがWである病棟が病院Hの中にある.
- 市民Cはまだ患者として受け入れられていない.
- 病棟Wには空きベッドがある.

admit(C:Cit, W:WID, H:Hos)によって

- 病院Hの病棟Wの属性である患者IDのリストに新しい患者のIDが加えられ，カルテが生成される.
- 病院のHの患者の集合に新しい患者が加わる.
- ある医療スタッフが新しい患者を受け持つことになる.

OTSモデル：遷移演算admitのCafeOBJ記述

```
op c-admit : Cit Wid Hos -> Bool
var C : Cit var W : W var H : Hos
eq c-admit(C, W, H) =  事前条件 EC
eq h-id(admit(C, W, H)) = h-id(H) .
eq h-loc(admit(C, W, H)) = h-loc(H) .
ceq wset(admit(C, W, H)) =  admit 後の病棟の集合
                            if c-admit(C, W, H) .
eq orset(admit(C, W, H)) = orset(H) .
eq phset(admit(C, W, H)) = phset(H) .
eq asset(admit(C, W, H)) = asset(H) .
ceq paset(admit(C, W, H)) =  admit 後の患者の集合
                             if c-admit(C, W, H) .
ceq msset(admit(C, W, H)) =  admit 後の医療スタッフの集合
                             if c-admit(C, W, H) .
eq cit-local(C, admit(C, W, H)) = cit-local(C, H) .
eq p-cond(P:Pid, admit(C, W, H)) = p-cond(P, H) .
ceq admit(C, W, H) = H if not(c-admit(C, W, H)) .
```

OTSモデル：遷移演算cch(citizen comes to a hospital)

- 遷移演算の定義の例2(cch: a citizen comes to a hospital.)
- cch(C:Cit, H:hos)の事前条件は市民Cが病院Hの外にいること.
- この事象の後，市民Cは病院の中に入る.

```
-- a citizen cames to a hospital.
eq c-cch(C, H1) = (c-loc(C) = outH) .
eq h-id(cch(C, H1)) = h-id(H1) .
eq h-loc(cch(C, H1)) = h-loc(H1) .
eq wset(cch(C, H1)) = wset(H1) .
eq orset(cch(C, H1)) = orset(H1) .
eq phset(cch(C, H1)) = phset(H1) .
eq asset(cch(C, H1)) = asset(H1) .
eq paset(cch(C, H1)) = paset(H1) .
eq msset(cch(C, H1)) = msset(H1) .
ceq cit-loc(Cid, cch(C, H1)) = (if (Cid = c-id(C)) then c-loc(in-h(C))
                                 else cit-loc(Cid, H1) fi)
                               if c-cch(C, H1) .
eq p-cond(Pid, cch(C, H1)) = p-cond(Pid, H1) .
ceq cch(C, H1) = H1 if not(c-cch(C, H1)) .
```

振舞いの例

- 振舞い
  - a citizen comes to a hospital,
  - admit,
  - interview,
  - plan analysis
- 上記の振舞いは右のようにCafeOBJの項として表すことができる.
- pmrid, wid, cはそれぞれ任意のカルテのID, 病棟のID, 市民を表す定数.

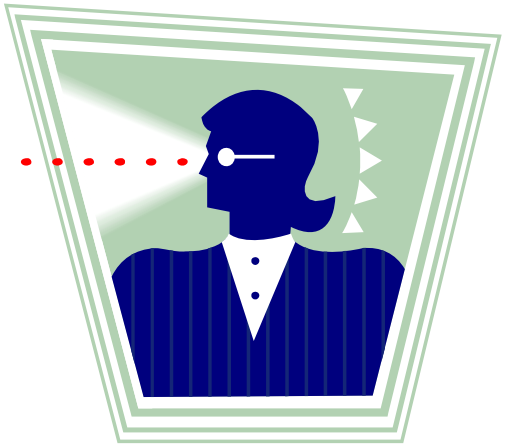
```
planAnal(pmrid,
         wid,
         interview(pmrid,
                   wid,
                   admit(c,
                         wid,
                         cch(c, init))))))))))
```

形式検証の例

- CafeOBJ仕様はCafeOBJ処理系の上で実行可能であり，その実行は形式記述(formal specification)の意味を規定する等式推論に忠実である
- CafeOBJ処理系を用いて，たとえば，次のような性質が証明できる
  - 病院の到達可能状態において
    - （性質１）病院のメレオロジーは常に成立つ
    - （性質２）ある病棟のベッドの数は常に患者の数よりも多い

（到達可能状態とは初期状態から任意の遷移演算を0回以上適用して到達できる状態）

まとめと今後の課題



汎用的記述 vs. 目的別記述

general formal description vs. specific formal description

documentation vs. analysis/verification

for human beings vs. for machines

formal methods vs. light-weight formal methods

一般的な記述はコストが高い

検証のためにも一般的な要件の洗い出しが必要になる
-- どの要件が検証に関係しているかは明らかではない

ドメインの記述における確認と検証

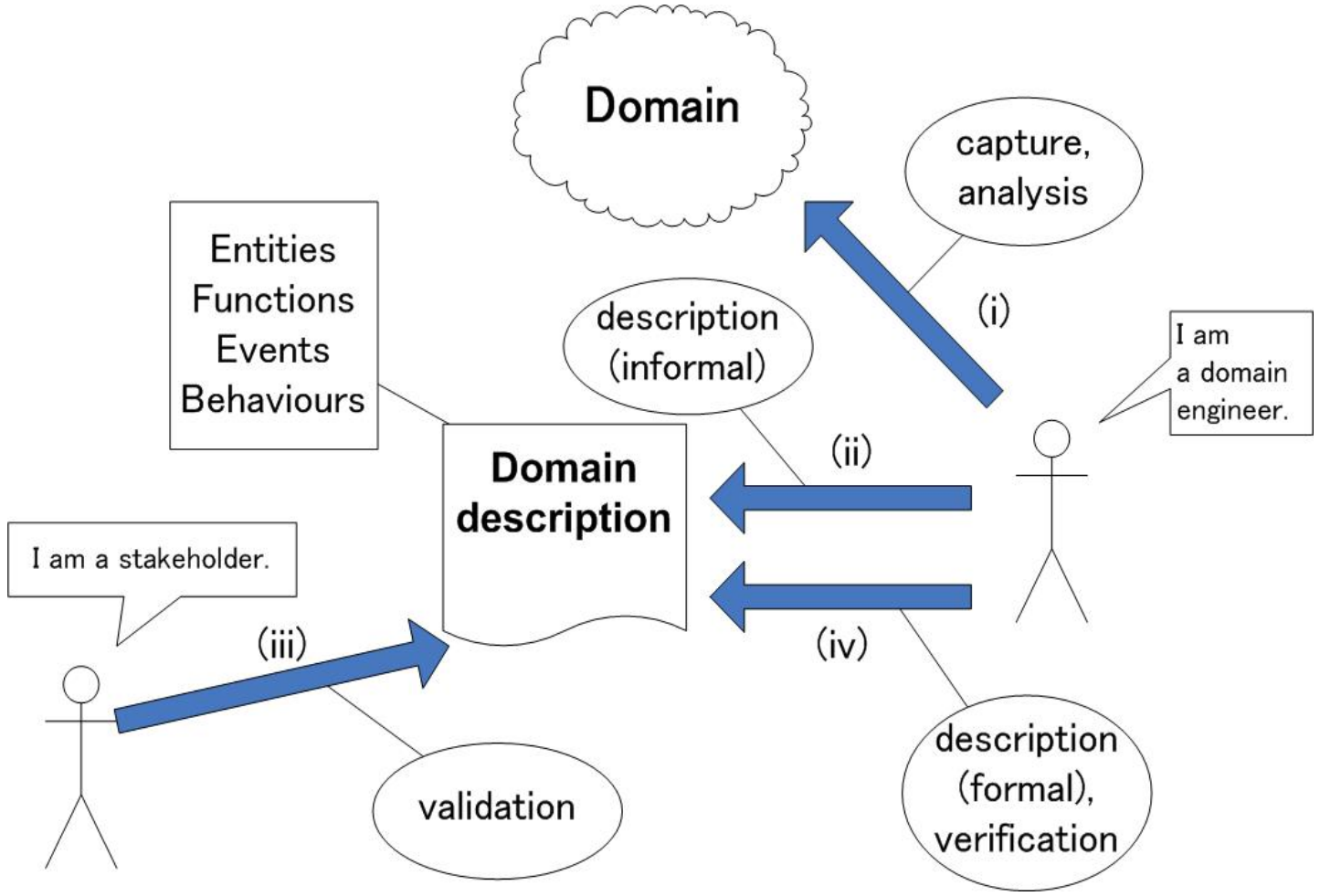
確認(validation)とは，ドメイン記述が利害関係者(stake holder)の意図と一致しているか確かめる作業，確認により「正しいドメイン記述」(the right domain description)を得る.

検証(verification)とは，ドメイン記述が記述者の意図するものと一致していることを証明すること，検証により「正しくドメイン記述」(the domain description right)を行う.

確認は図や自然言語などで記述された非形式ドメイン記述を利害関係者に提示することによって行われ，検証は形式記述がドメインが持つべき性質を満たしていることを形式的に証明することによって行われる.

確認と検証は補完的なものであり，確認により形式記述が改良され検証が促進され，検証により非形式記述が改良され確認が促進される.

ドメインの形式記述の理想的手順



ライセンス言語の考え方とその可能性

- ライセンスは，ドメインの実体が動作（関数）を実行し得る条件を明確にすることで，ドメインの振舞を規定する
- ライセンス言語は，権利を有する実体が権利を行使する実体にその権利の行使権（ライセンス）を保証することを記述する
- ライセンスとライセンス言語は，法令，規則，プロトコルなどで規定されるドメインにおける振舞の記述とその性質の解析において，有効であると予想される

ライセンス記述の例：

License i : licensor a grants licence b on work w
with permitted actions {a1,a2,a3,…,am}
and obligated actions {b1,b2,b3,…,bn}