はてなブックマークにおけるアクセス制御

半環構造に基づくモデル化

伊奈 林太郎

id:tarao @oarat

2015-11-24

@ WebDB Forum 2015


B!
10th ANNIVERSARY

- 国内最大のソーシャルブックマークサービス

- 国内最大のソーシャルブックマークサービス
- Scala でいちから作り直すプロジェクトが進行中

DISCLAIMER

- 内容は開発中のもの
  - 実際のリリース時には変更の可能性あり
  - 最終的にどうなったか公表するかどうか未定

話すこと

- B!のアクセス制御はかなり複雑
  - さまざまな画面/設定
  - AND/OR 条件
- 統一的なモデル化でバグの作り込みを避けたい
  - モデルの数学的な正しさ
  - コード上で一貫して扱うしくみ
- 効率化したい
  - DB クエリ等で権限に応じたフィルタリング

アクセス制御のモデル化

単純なアクセス制御

- 非公開ユーザの設定
- ブックマークが他人からは見えなくなる
- 本人にはもちろん見える

単純なアクセス制御

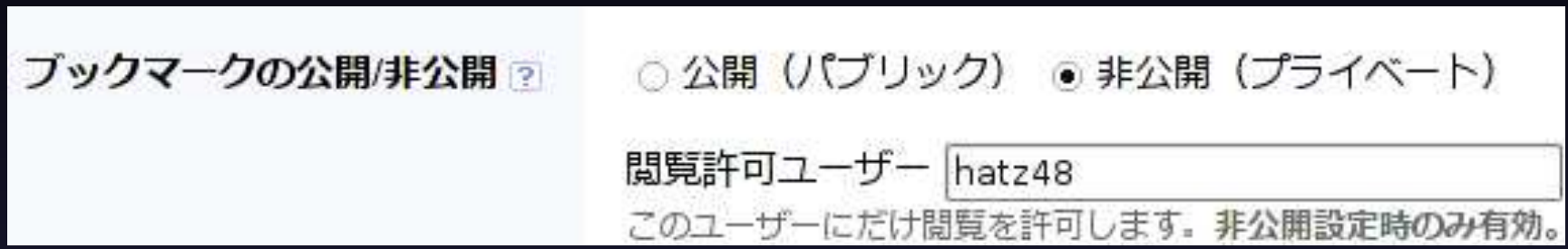
- 非公開ユーザの設定
- ブックマークが他人からは見えなくなる
- 本人にはもちろん見える

コード

```
if (bookmark.owner == visitor)
  Some(bookmark)
else None
```

少し複雑なアクセス制御

- 閲覧許可ユーザの設定
- 本人に加えて許可された人にも見える



少し複雑なアクセス制御

- 閲覧許可ユーザの設定
- 本人に加えて許可された人にも見える

愚直なコード

```
if (bookmark.owner == visitor ||
    bookmark.owner.allowedUsers.contains(visitor))
  Some(bookmark)
else None
```

少し複雑なアクセス制御

- 閲覧許可ユーザの設定
- 本人に加えて許可された人にも見える

愚直なコード

```
if (bookmark.owner == visitor ||
    bookmark.owner.allowedUsers.contains(visitor))
  Some(bookmark)
else None
```

- 少し複雑
- そのうちチェックしわすれが発生

少し複雑なアクセス制御

- 閲覧許可ユーザの設定
- 本人に加えて許可された人にも見える

愚直なコード

```
if (bookmark.owner == visitor ||
    bookmark.owner.allowedUsers.contains(visitor))
  Some(bookmark)
else None
```

- 少し複雑
- そのうちチェックしわすれが発生

なにか統一的に扱いたい

アクセス制御のモデル

属性 A の冪で表現

- 要求 $r \in \mathcal{P}(A) = \texttt{requestOf}(viewer)$
- 許可 $p \in \mathcal{P}(A) = \texttt{permissionOf}(target)$
- 確認 $p \,\texttt{allows}\, r \;\Leftrightarrow\; p \cap r \neq \emptyset$

アクセス制御のモデル

属性 A の冪で表現

- 要求 $r \in \mathcal{P}(A) = \texttt{requestOf}(viewer)$
- 許可 $p \in \mathcal{P}(A) = \texttt{permissionOf}(target)$
- 確認 $p\ \texttt{allows}\ r \Leftrightarrow p \cap r \neq \emptyset$

例

ユーザ	閲覧許可	要求	許可
u_1	なし	$\{u_1\}$	$\{u_1\}$
u_2	u_1, u_3	$\{u_2\}$	$\{u_1, u_2, u_3\}$
u_3	u_2	$\{u_3\}$	$\{u_2, u_3\}$

$\texttt{permissionOf}(u_1)\ \texttt{allows}\ \texttt{requestOf}(u_2) = \texttt{false}$

$\texttt{permissionOf}(u_2)\ \texttt{allows}\ \texttt{requestOf}(u_3) = \texttt{true}$

$\texttt{permissionOf}(u_3)\ \texttt{allows}\ \texttt{requestOf}(u_1) = \texttt{false}$

アクセス制御のモデル

属性 A の冪で表現

要求 $r \in \mathcal{P}(A) = \texttt{requestOf}(viewer)$

許可 $p \in \mathcal{P}(A) = \texttt{permissionOf}(target)$

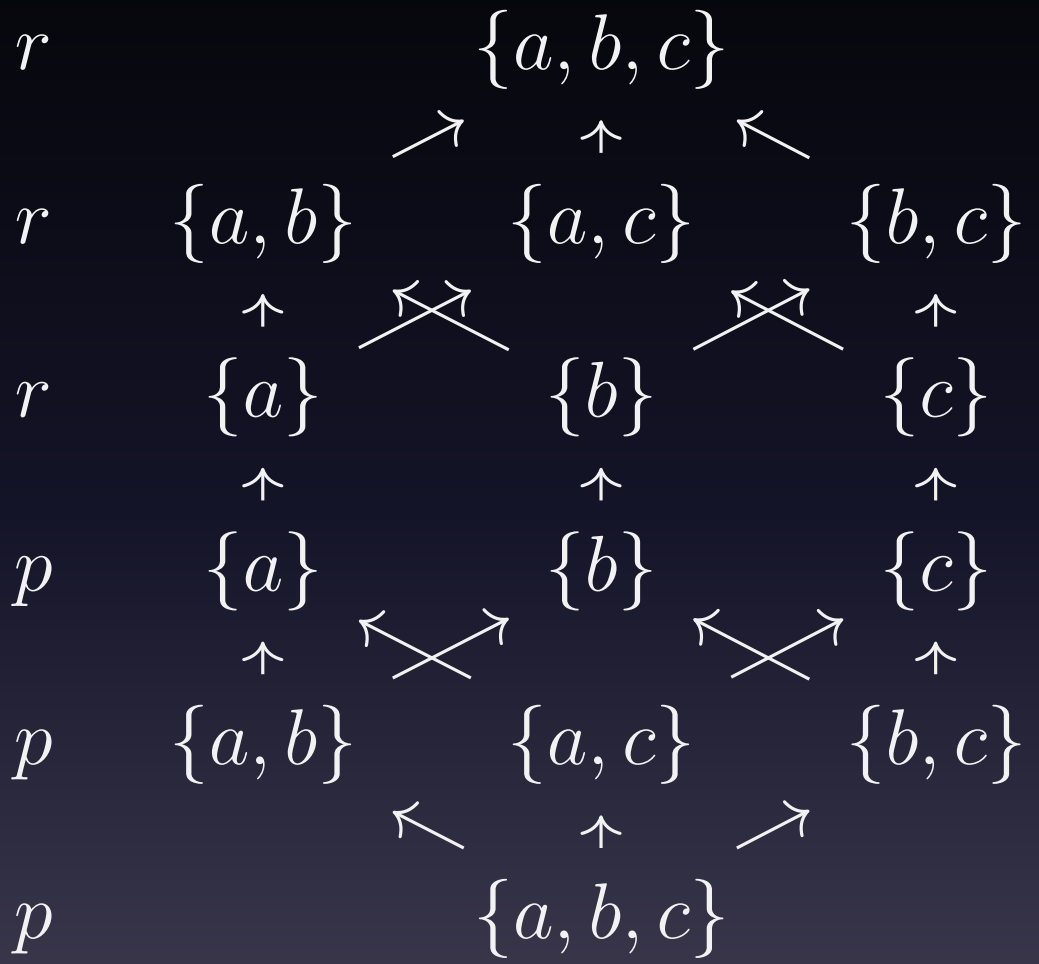
確認 $p\ \texttt{allows}\ r \Leftrightarrow p \cap r \neq \emptyset$

コードによる表現

```
val r = requestOf(visitor)
val p = permissionOf(bookmark.owner)
if (p allows r) Some(bookmark)
else None
```

- オブジェクトから属性集合へのマッピング (permissionOf, requestOf)
- オブジェクト型ごとの一貫したメンテが可能

アクセス制御のモデル



- Lattice-based access control (LBAC)
- 権限の強弱関係が束をなす

もっと複雑なアクセス制御

- 個別のブックマークを非公開にできる
- 閲覧許可ユーザにも見えない

もっと複雑なアクセス制御

- 個別のブックマークを非公開にできる
- 閲覧許可ユーザにも見えない

愚直なコード

```
if (bookmark.owner == visitor ||
    (bookmark.owner.allowedUsers.contains(visitor) &&
     bookmark.isPublic))
  Some(bookmark)
else None
```

もっと複雑なアクセス制御

- 個別のブックマークを非公開にできる
- 閲覧許可ユーザにも見えない

愚直なコード

```
if (bookmark.owner == visitor ||
    (bookmark.owner.allowedUsers.contains(visitor) &&
     bookmark.isPublic))
  Some(bookmark)
else None
```

- LBAC では表現不能

もっと複雑なアクセス制御

- 個別のブックマークを非公開にできる
- 閲覧許可ユーザにも見えない

愚直なコード

```
if (bookmark.owner == visitor ||
    (bookmark.owner.allowedUsers.contains(visitor) &&
     bookmark.isPublic))
  Some(bookmark)
else None
```

- LBAC では表現不能

(※実際はあと 5 段階くらい複雑)

もっと複雑なアクセス制御

- 個別のブックマークを非公開にできる
- 閲覧許可ユーザにも見えない

愚直なコード

```
if (bookmark.owner == visitor ||
    (bookmark.owner.allowedUsers.contains(visitor) &&
     bookmark.isPublic))
  Some(bookmark)
else None
```

- LBAC では表現不能

(※実際はあと 5 段階くらい複雑)

もっとよいモデル化が必要

アクセス制御のよいモデル

属性 A の半環で許可を表現

- 要求 $r \in \mathcal{P}(A) = \texttt{requestOf}(viewer)$
- 許可 $p \in \mathcal{P}(\mathcal{P}(A)) = \texttt{permissionOf}(target)$
- 確認 $p\ \texttt{allows}\ r \ \Leftrightarrow\ \exists q \in p.\ q \subseteq r$
- 半環 $\langle \mathcal{P}(\mathcal{P}(A)), \oplus, \otimes, \emptyset, \{\emptyset\} \rangle$
  - $p_1 \oplus p_2 = p_1 \cup p_2$
  - $p_1 \otimes p_2 = p_1 \Cup p_2 = \{q_1 \cup q_2 \mid q_1 \in p_1 \wedge q_2 \in p_2\}$

アクセス制御のよいモデル

属性 A の半環で許可を表現

- 要求 $r \in \mathcal{P}(A) = \texttt{requestOf}(viewer)$
- 許可 $p \in \mathcal{P}(\mathcal{P}(A)) = \texttt{permissionOf}(target)$
- 確認 $p \texttt{ allows } r \;\Leftrightarrow\; \exists q \in p.\, q \subseteq r$
- 半環 $\langle \mathcal{P}(\mathcal{P}(A)), \oplus, \otimes, \emptyset, \{\emptyset\} \rangle$
  - $p_1 \oplus p_2 = p_1 \cup p_2$
  - $p_1 \otimes p_2 = p_1 \Cup p_2 = \{q_1 \cup q_2 \mid q_1 \in p_1 \wedge q_2 \in p_2\}$

直感

- $p = \{\{a_1, a_2\}, \{a_3\}, \{a_4, a_5\}\}$
- $(a_1$ かつ $a_2)$ または a_3 または $(a_4$ かつ $a_5)$ の属性

アクセス制御のよいモデル

属性 A の半環で許可を表現

- 要求 $r \in \mathcal{P}(A) = \texttt{requestOf}(viewer)$
- 許可 $p \in \mathcal{P}(\mathcal{P}(A)) = \texttt{permissionOf}(target)$
- 確認 $p\ \texttt{allows}\ r \ \Leftrightarrow\ \exists q \in p.\ q \subseteq r$
- 半環 $\langle \mathcal{P}(\mathcal{P}(A)), \oplus, \otimes, \emptyset, \{\emptyset\} \rangle$
  - $p_1 \oplus p_2 = p_1 \cup p_2$
  - $p_1 \otimes p_2 = p_1 \Cup p_2 = \{q_1 \cup q_2 \mid q_1 \in p_1 \wedge q_2 \in p_2\}$

コードによる表現

```
val r = requestOf(visitor)
val p1 = permissionOf(bookmark.owner)
val p2 = permissionOf(bookmark)
val p = p1 & p2 // p1 ⊗ p2
if (p allows r) Some(bookmark)
else None
```

半環構造の妥当性

もしこう書いてしまったら?

```
val r = requestOf(visitor)
val p1 = permissionOf(bookmark.owner)
val p2 = permissionOf(bookmark)
if (p1 allows r) {
  if (p2 allows r) Some(bookmark)
  else None
} else None
```

半環構造の妥当性

Theorem (ブール代数への準同型写像)

$\forall r.$ _ `allows` $r : \mathcal{P}(\mathcal{P}(A)) \to \mathbb{B}$ は準同型写像

もしこう書いてしまったら? $\to$ OK!

```
val r = requestOf(visitor)
val p1 = permissionOf(bookmark.owner)
val p2 = permissionOf(bookmark)
if (p1 allows r) {
  if (p2 allows r) Some(bookmark)
  else None
} else None
```

半環構造の妥当性

Theorem (ブール代数への準同型写像)

$\forall r.\ \_\ \texttt{allows}\ r : \mathcal{P}(\mathcal{P}(A)) \to \mathbb{B}$ は準同型写像

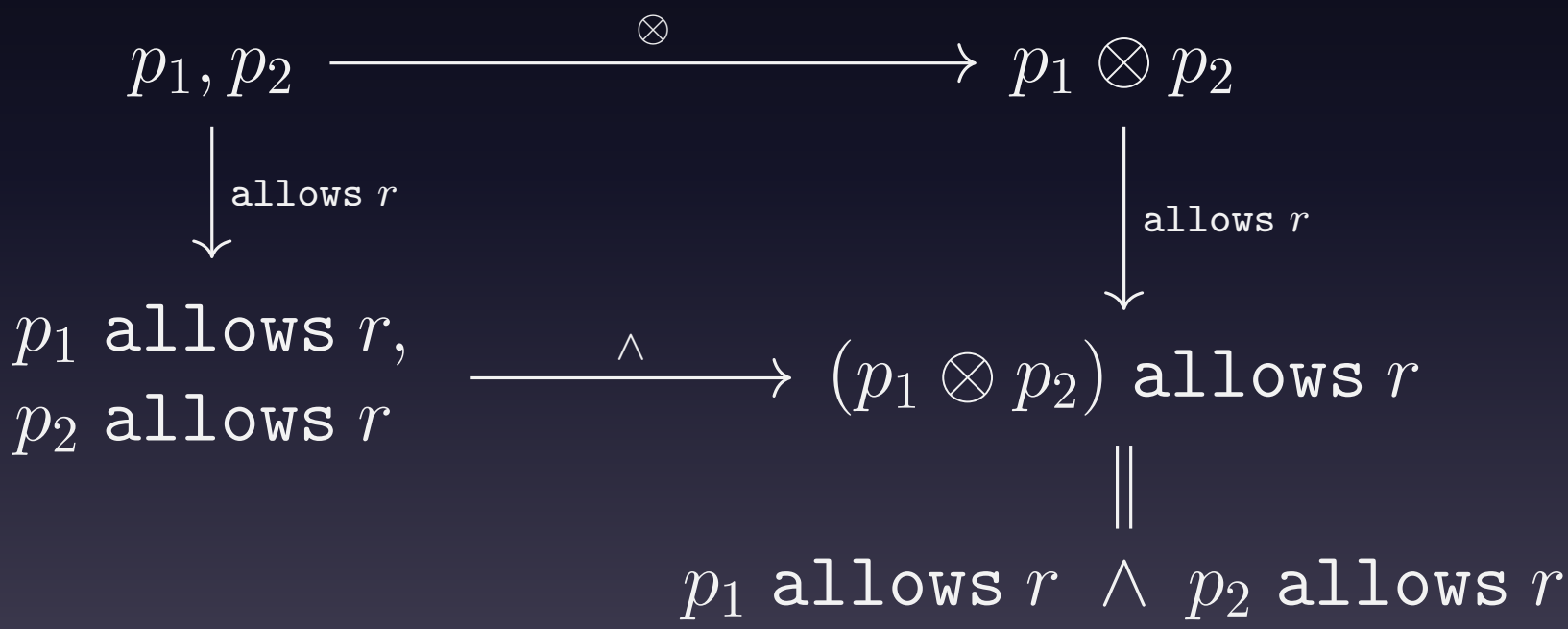
注意

- $\langle \mathbb{B}, \vee, \wedge, \bot, \top \rangle$: ブール代数
- $p\ \texttt{allows}\ r : \mathbb{B}$
- $\_\ \texttt{allows}\ \_ : \mathcal{P}(\mathcal{P}(A)) \to \mathcal{P}(A) \to \mathbb{B}$
- $\_\ \texttt{allows}\ r : \mathcal{P}(\mathcal{P}(A)) \to \mathbb{B}$　(部分適用)

半環構造の妥当性

Theorem (ブール代数への準同型写像)

$\forall r.\ \_\ \texttt{allows}\ r : \mathcal{P}(\mathcal{P}(A)) \to \mathbb{B}$ は準同型写像



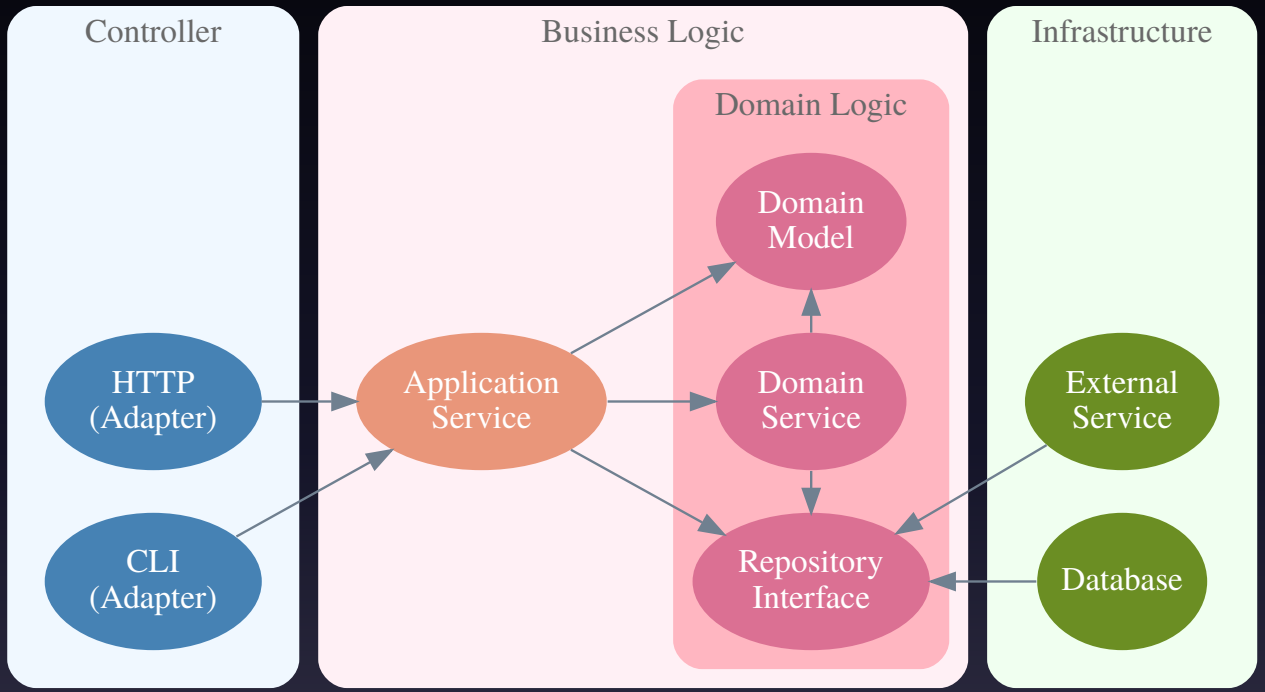
- 条件を分解してチェックしても OK

出典

- G. Karvounarakis and T. J. Green.
  Semiring-annotated data: Queries and provenance?
  SIGMOD Record, 41(3):5–14, 2012.
- R. Thion, F. Lesueur, and M. Talbi.
  Tuple-based access control: a provenance-based information flow control for relational data.
  In *30th ACM/SIGAPP SAC*, 2015.

実装戦略

基本戦略



複数層に権限チェックを分離可能 (∵準同型定理)

- アプリ層：前提条件でフィルタ
- ドメイン層：完全な権限チェック
- インフラ層：効率化のためのフィルタ

各層でのチェック

アプリ層：前提条件でフィルタ

- (例) 非公開ユーザのページは他人に見せない

ドメイン層：完全な権限チェック

- ⊗ された完全な権限をチェック
- アクセス制御の完全性はこの層に集中させる

インフラ層：効率化のためのフィルタ

- 権限の要求を検索クエリに対応させる
- 各テーブルごとでよい
- 完全でなくてよい

整合性のための工夫

一貫性

- 属性へのマッピングは一箇所で定義
- 同じ型には同じ方法で要求/許可が決まる

```
def requestOf(user: User): Request = user.accountId match {
  case AccountId.Guest => Request(Attribute.Public)
  case _               => Request(
    Attribute.Public,
    Attribute.User(user.id)) }

def permissionOf(user: User): Permission = user.status match {
  case User.Status.Private => Permission(
    Attribute.User(user.id) +:
    user.allowedUsers.map(u => Attribute.User(u.id)): _*)
  case User.Status.Public  => Permission(
    Attribute.Public,
    Attribute.User(user.id)) }
```

整合性のための工夫

完全性

- 許可の可否のための完全な情報を使用
  - (例) ブックマークの閲覧にはユーザの閲覧許可も必要

```
// ユーザ閲覧可否なしにブックマーク閲覧可否だけ調べるのは禁止
private def permissionOf(bookmark: Bookmark): Permission = ...

// ブックマーク閲覧可否は以下の 2 つの合成
// - 持ち主のユーザ閲覧可否
// - ブックマークアイテムそのものの閲覧可否
def permissionOf(pair: (User, Bookmark)): Permission =
  permissionOf(user = pair._1) &   // ユーザ
  permissionOf(bookmark = pair._2) // ブックマークそのもの
```

検索クエリとの対応

クエリとの対応

- インフラ層での MySQL や Elasticsearch のクエリ
- 許可されないレコードは除いて効率化
- 要求 $\rightarrow$ クエリ断片への変換を用意すればよい

```
val r = requestOf(visitor)
// => Request(Attribute.User(u))

val user: Option[User] = db.run { sql"""
  SELECT * FROM user
    LEFT JOIN allowing
    ON user.id = allowing.user_id
  WHERE user.status = 'public'
     OR user.id = $u
     OR allowing.allowed_user_id = $u
""".as[User] }.headOption
```

完全 vs. テーブルごと

`permissionOf(user) & permissionOf(bookmark)`

- JOIN が必要
  - user テーブルと bookmark テーブル
- 条件が複雑
  - AND 条件と OR 条件

テーブルごとにチェック

- bookmark テーブルの検索時は `permissionOf(bookmark)` だけチェック
- `permissionOf(user)` は別の層でチェック
- 準同型定理が活かされる

その他の効率化

和積形

許可は和積形 p' でも表せる

- `user`：本人 $\oplus$ 閲覧許可ユーザ
- `bookmark`：本人 (非公開の場合)
- 全体: (本人 $\oplus$ 閲覧許可ユーザ) $\otimes$ (本人)

和積形のままチェック

- p' `allows` $r \Leftrightarrow \forall q \in p'.\, q \cap r \neq \emptyset$
- 積和形のチェックと同値
  p' `allows` $r \Leftrightarrow p$ `allows` r

和積形

許可は和積形 p' でも表せる

- user : 本人 $\oplus$ 閲覧許可ユーザ
- bookmark : 本人 (非公開の場合)
- 全体: (本人 $\oplus$ 閲覧許可ユーザ) $\otimes$ (本人)

和積形のままチェック

- p' allows $r \Leftrightarrow \forall q \in p'.\, q \cap r \neq \emptyset$
- 積和形のチェックと同値
  p' allows $r \Leftrightarrow p$ allows r

$\otimes$ 時に $\cup$ を計算する必要はない

まとめ

モデル化

- AND/OR 条件を属性の半環で表現
- チェック関数はブール代数への準同型写像

実装戦略

- 準同型定理を利用してチェック処理を分離
- 要求/許可は一元管理
- 完全性の担保はドメインロジックに集中

効率化

- 要求をテーブルへのクエリ条件にしてフィルタ
- 和積形の利用で無駄な計算をしないように