SHINSHU UNIVERSITY

# 上流設計からモデル検査プロセスまでの一貫設計検証環境


宮本 直樹† 和崎 克己††

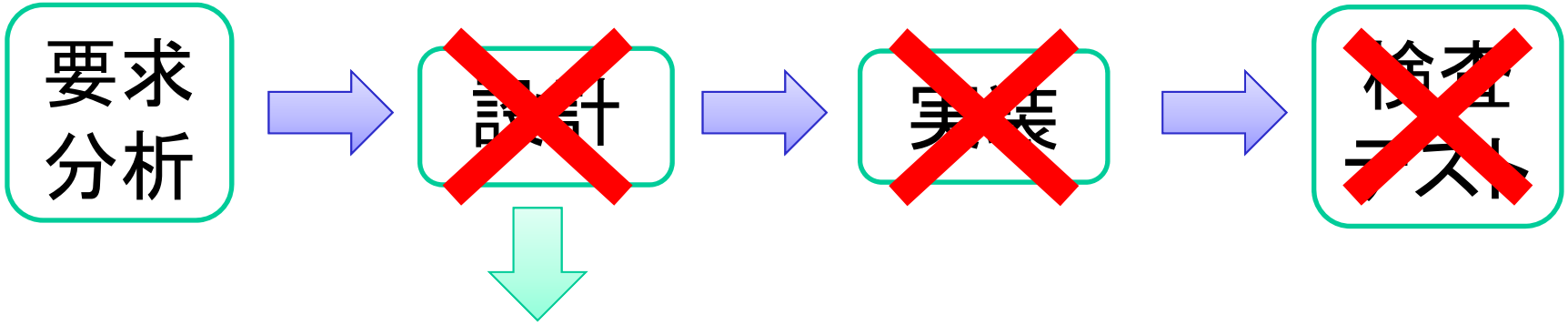
†信州大学大学院 工学系研究科 情報工学専攻

††信州大学工学部 情報工学科

# 目次

# 背景

システムの開発プロセス




モデルの性質を自動で検証する
モデル検査が注目されている

設計段階での欠陥を抽出することができ
ソフトウェアの品質を飛躍的に向上できる

# 研究の概要

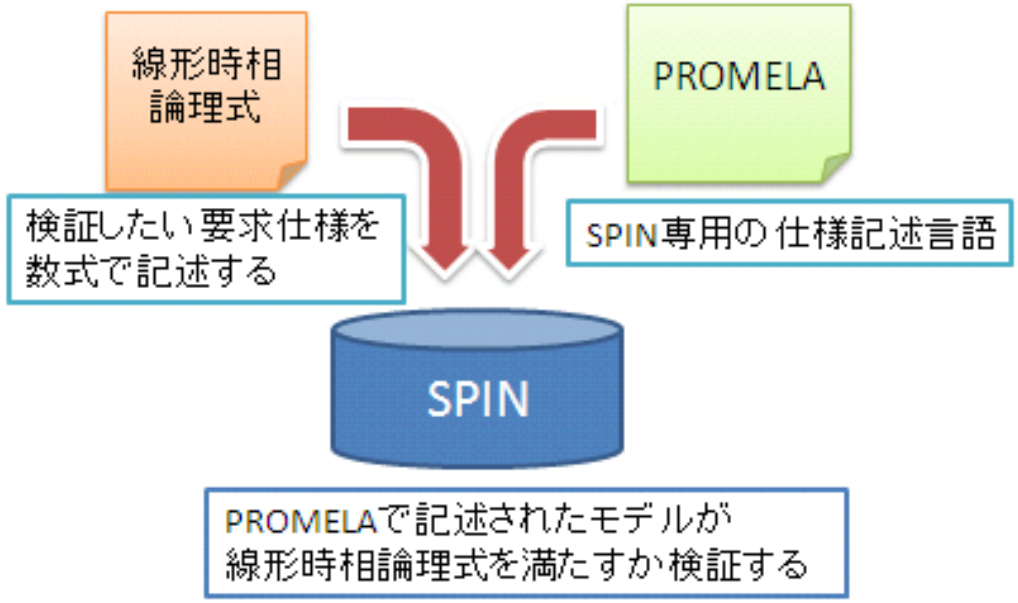
UML図で検査対象の
モデルの上流設計を記述

SPINモデル検査向けの
プロセス定義へ自動変換

上流設計から，モデル検査プロセスまでの
一貫した設計検証環境を提供する

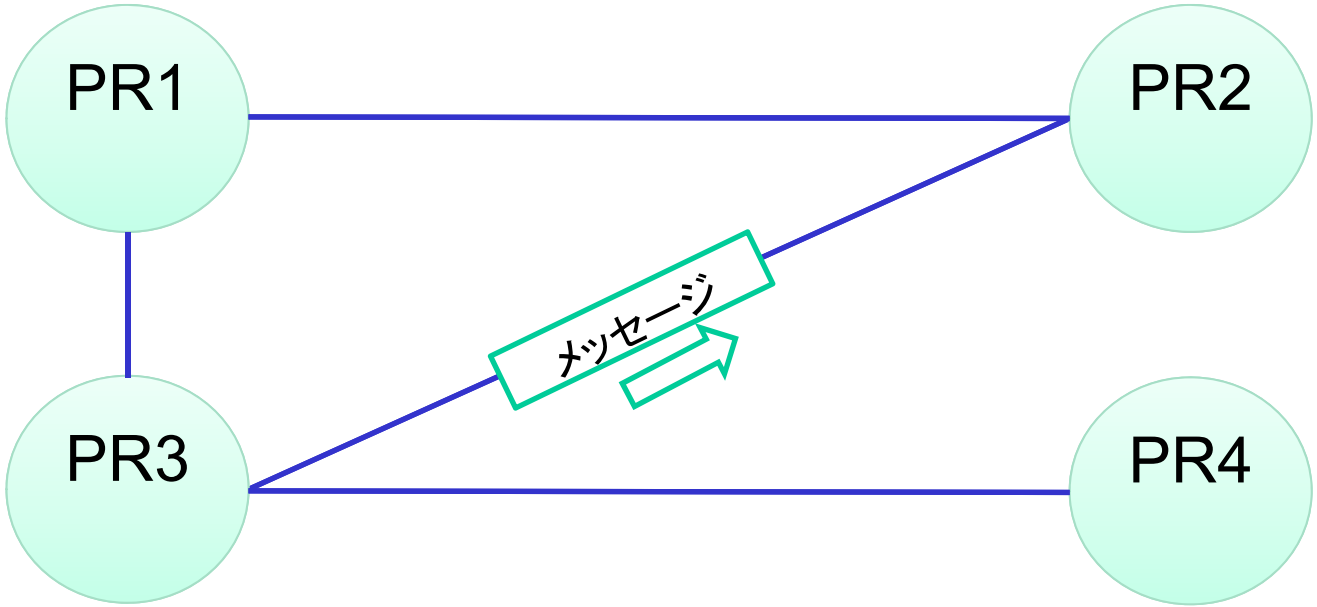
# SPINモデル検査器

- モデル検査器の一種
- 仕様記述言語PROMELAで記述されたモデルの形式検証が行える
- 線形時相論理(LTL)式で要求仕様を記述することで，その仕様が正しいかどうかを検証できる



# PROMELA

- モデル検査器SPINで用いられる仕様記述言語
- プロセス毎に並行動作するシステムや，非決定的な振る舞いを容易に記述できる
- チャネルを用いて，相互通信のシステム検証に使われることが多い

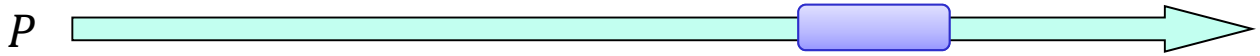


# LTL(Linear Temporal Logic)式とは

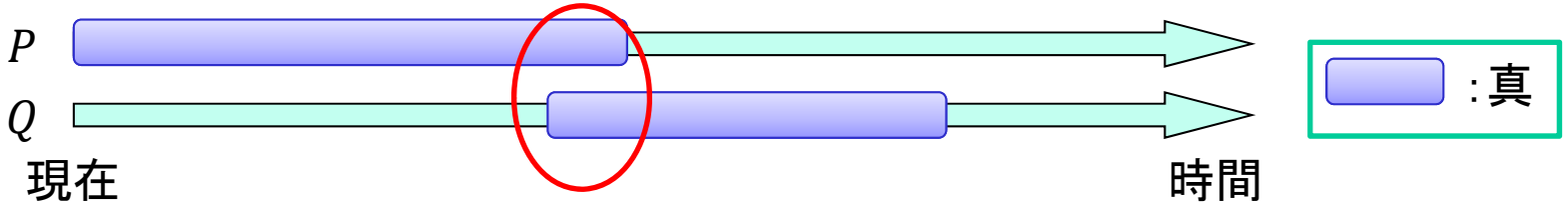
- 時相論理の一種
- 時間が一直線に進む性質を記述する
- SPINは，PROMELAで記述されたモデルがLTL式を満たすかどうか検証する機能を有する

□$P$：常に命題$P$が成り立つ


◇$P$：いつかは命題$P$が成り立つ

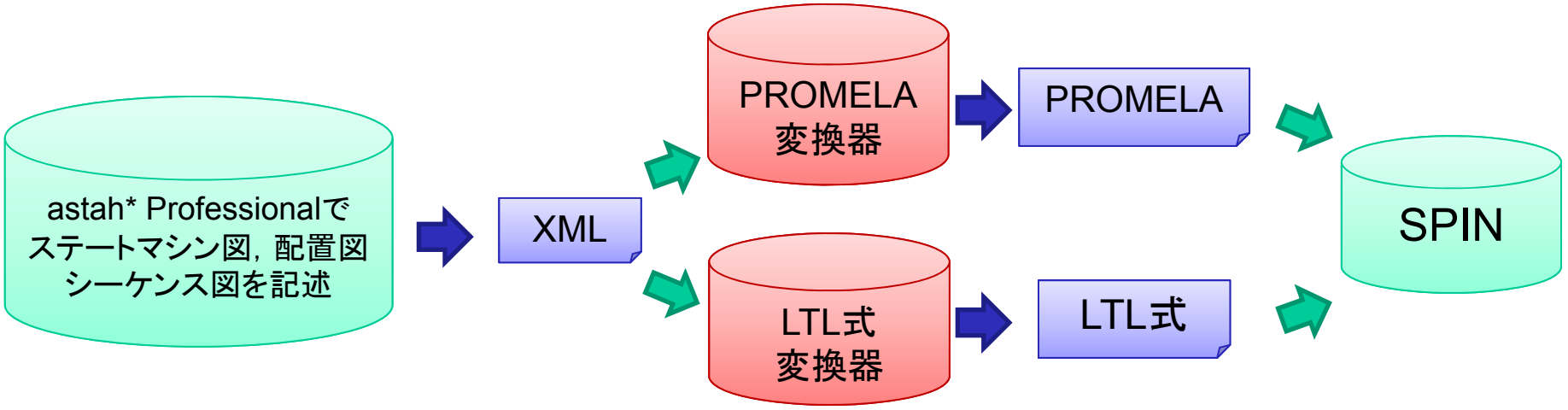


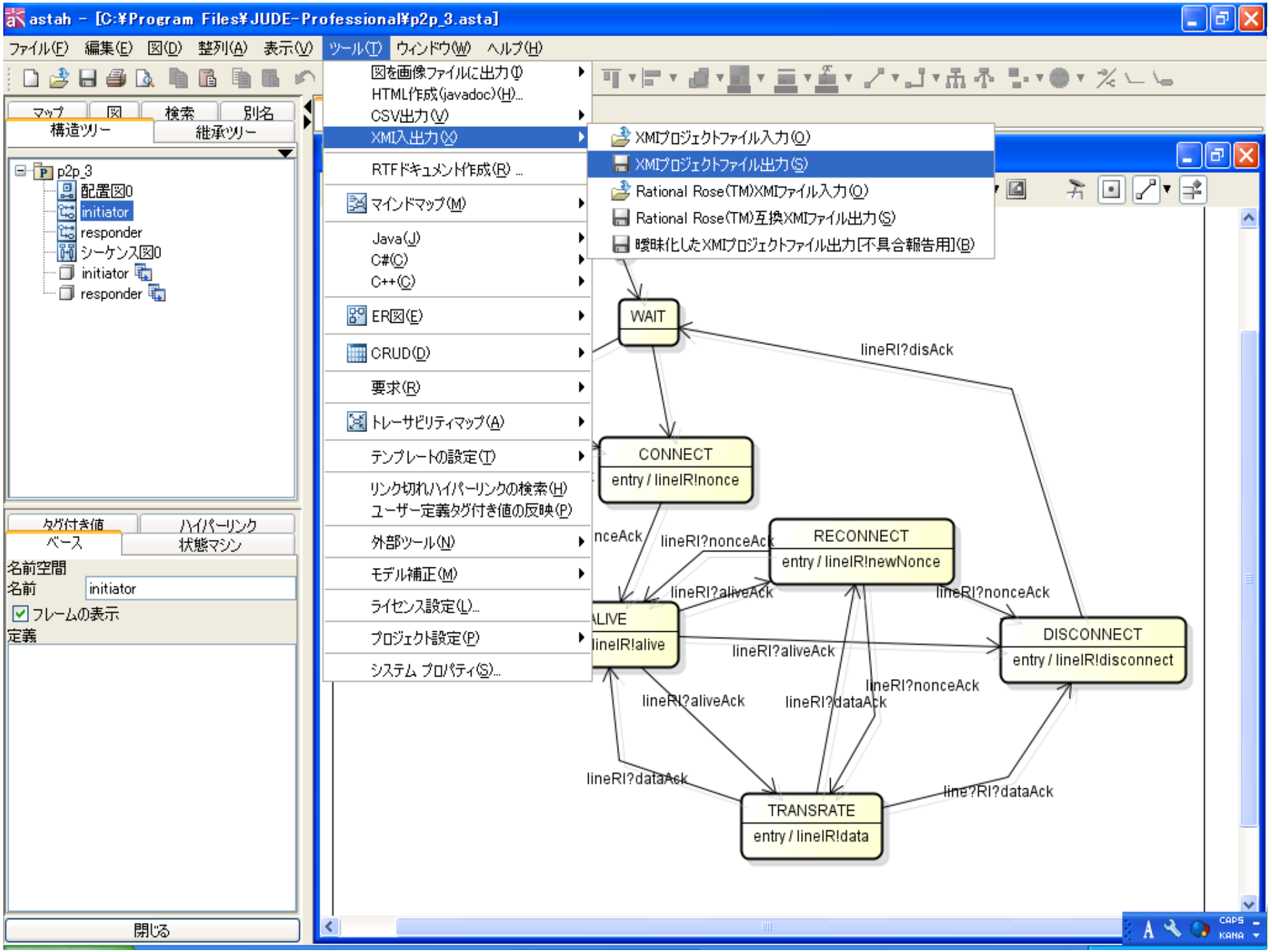

$P$ U $Q$：命題$Q$が真になるまで，$P$は真である



# 自動変換の流れ

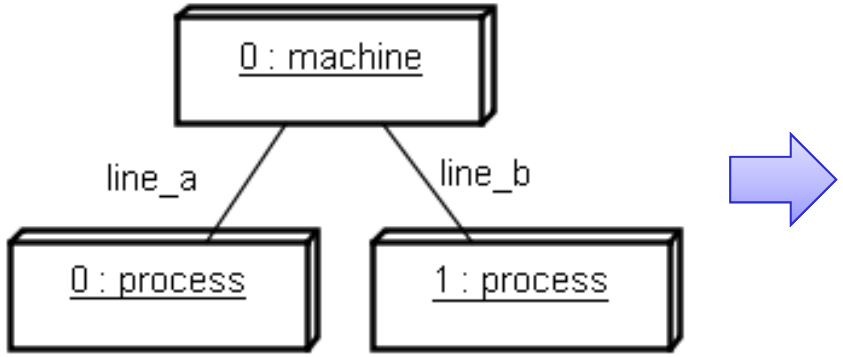
- UML記述ツール『astah* Professional』で，対象モデルを記述し，XML形式のファイルで出力
- ステートマシン図，配置図をPROMELAに変換
- シーケンス図を，LTL式に変換




astah - [C:¥Program Files¥JUDE-Professional¥p2p_3.asta]
ファイル(F) 編集(E) 図(D) 整列(A) 表示(V) ツール(T) ウィンドウ(W) ヘルプ(H)
図を画像ファイルに出力(I)
HTML作成(javadoc)(H)...
CSV出力(V)
XMI入出力(X)
RTFドキュメント作成(R) ...
マインドマップ(M)
Java(J)
C#(C)
C++(C)
ER図(E)
CRUD(D)
要求(R)
トレーサビリティマップ(A)
テンプレートの設定(T)
リンク切れハイパーリンクの検索(H)
ユーザー定義タグ付き値の反映(P)
外部ツール(N)
モデル補正(M)
ライセンス設定(L)...
プロジェクト設定(P)
システム プロパティ(S)...
XMIプロジェクトファイル入力(O)
XMIプロジェクトファイル出力(S)
Rational Rose(TM)XMIファイル入力(O)
Rational Rose(TM)互換XMIファイル出力(S)
曖昧化したXMIプロジェクトファイル出力[不具合報告用](B)
マップ 図 検索 別名
構造ツリー 継承ツリー
p2p_3
配置図0
initiator
responder
シーケンス図0
initiator
responder
タグ付き値 ハイパーリンク
ベース 状態マシン
名前空間
名前 initiator
フレームの表示
定義
閉じる
WAIT
lineRI?disAck
CONNECT
entry / lineIR!nonce
RECONNECT
entry / lineIR!newNonce
lineRI?nonceAck
lineRI?aliveAck
ALIVE
lineIR!alive
DISCONNECT
entry / lineIR!disconnect
lineRI?dataAck
TRANSRATE
entry / lineIR!data
line?RI?dataAck
CAPS
KANA

# 配置図の変換

- 配置図で，モデルの配置状況と，接続状況を記述
- 配置図で定義されたリンク線を，PROMELAのチャネルとする
- 配置図のインスタンス数を，PROMELAのプロセス数とする



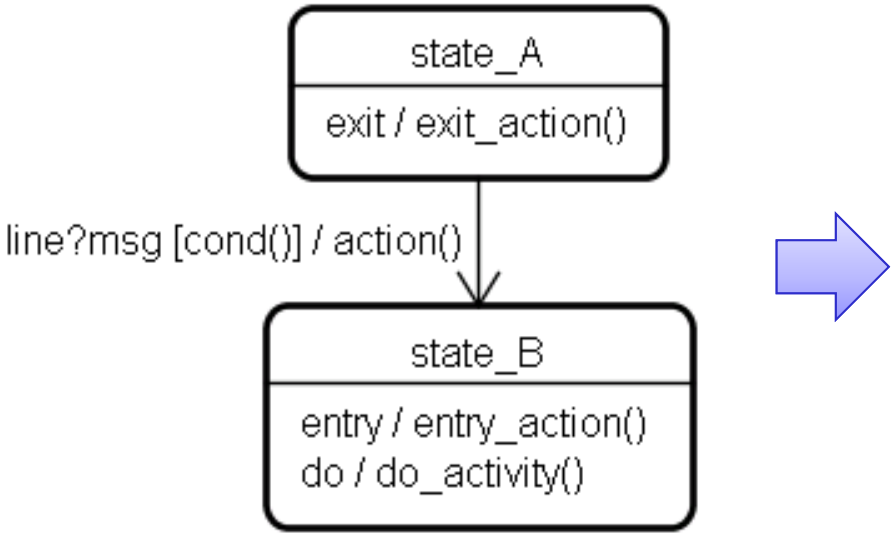

```
chan line_a = [0] of {mtype};
chan line_b = [0] of {mtype};
```

```
active proctype machine_0_state{
…
}
active proctype process_0_state{
…
}
active proctype process_1_state{
…
}
```

配置図のモデルの変換例

# ステートマシン図の変換

- 対象モデルの振る舞いを，状態遷移で記述
- コンポジット状態への対応

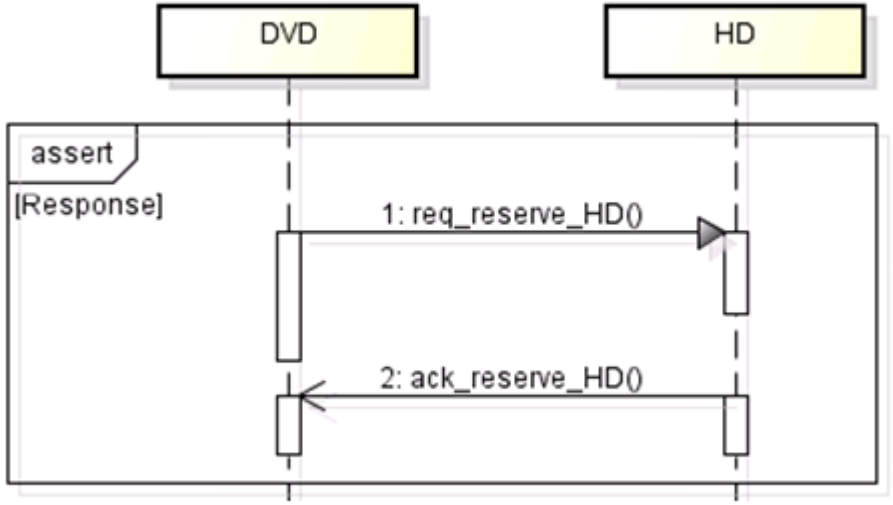



```
do
 :: line?event ->
    if
     :: machine_0_state == state_A
           && event == msg && cond() ->
        exit_action();
        machine_0_state = state_B;
        entry_action();
        do_activity();
    fi
od
```

ステートマシン図からPROMELAへの変換例

# UMLシーケンス図の変換

- シーケンス図に記述された構造を，LTL式へ変換する
- 用いる要素は，ライフライン，複合フラグメント，メッセージのみとする
- LTL式への変換は，仕様パターンに準拠する



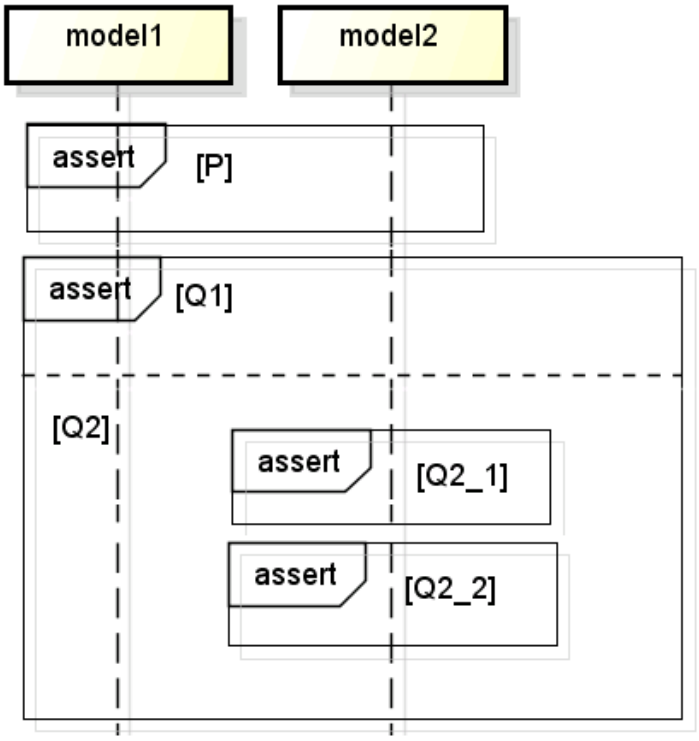
# 仕様パターン

- 開発の際によく用いられる要求仕様をパターン化し，時相論理式として与える
- 各パターンにおいて，適用範囲となる時間制約を表すスコープがある
  - Betweenスコープ
  - Afterスコープ
  - Beforeスコープ
  - After Untilスコープ

SPEC PATTERNS,
http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml.

# 複合フラグメント

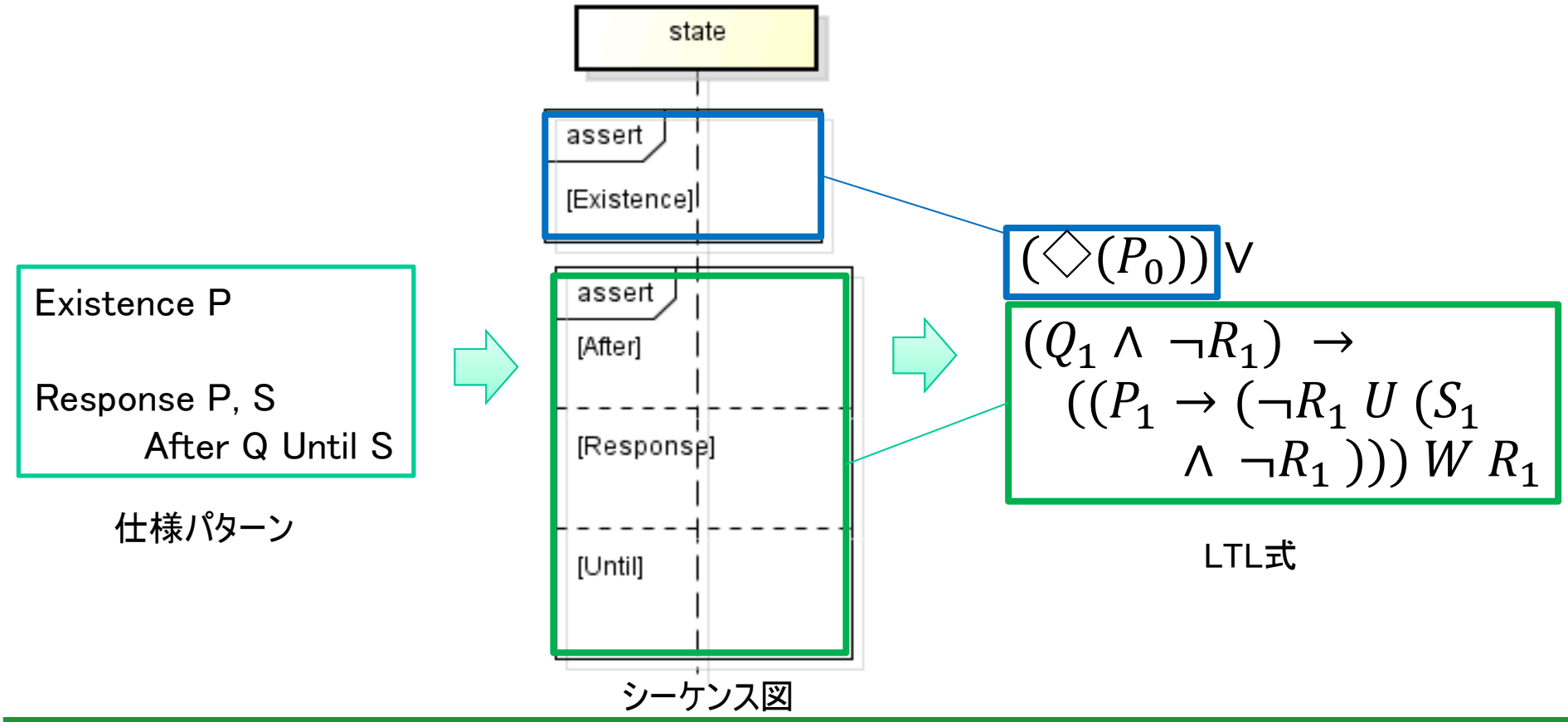
- 複数のフラグメントを論理演算子へ変換
- 論理式の記述性を向上




$$P \vee (Q1 \wedge (Q2_1 \vee Q2_2))$$

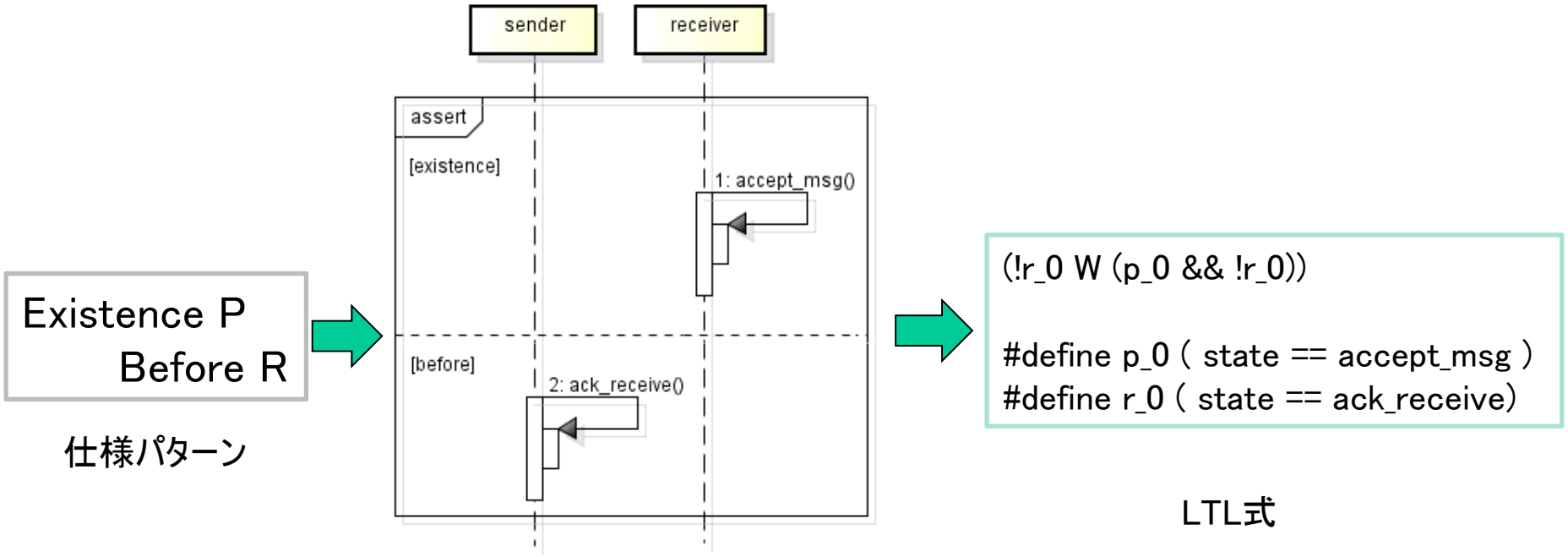
# 複合フラグメントと仕様パターンの対応

- 複合フラグメントの名前に，仕様パターン名，スコープ名を記述
- 複合フラグメントの名前からLTL式を取得する



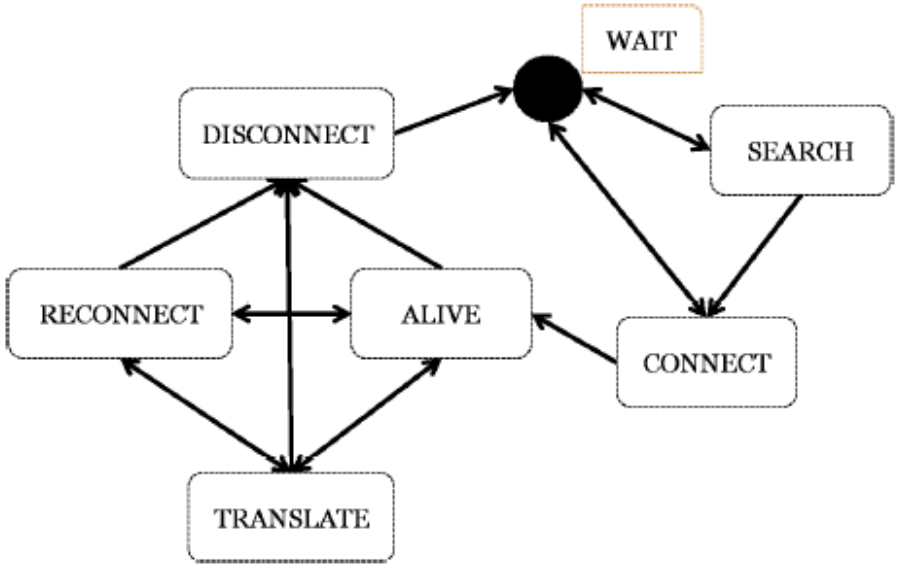
# メッセージと論理変数の対応

- シーケンス図のメッセージ要素の記述内容を，LTL式の論理変数と対応付ける
- 出力されたLTL式を用いて，SPINでモデル検査を実行する



# 変換例：永続化プロトコルの検証

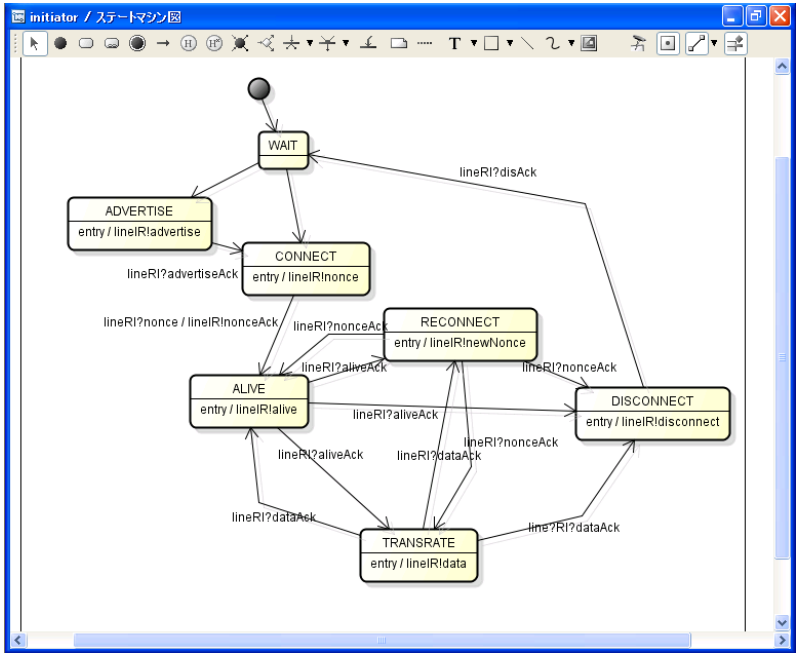
- 従来の通信プロトコルにRECONNECT状態とALIVE状態を追加
- 再接続状態でも通信が可能で，従来のプロトコルよりコストを抑えることができる
- 通信を制御するInitiatorと，データを受け取るResponder間での応答性を検証する



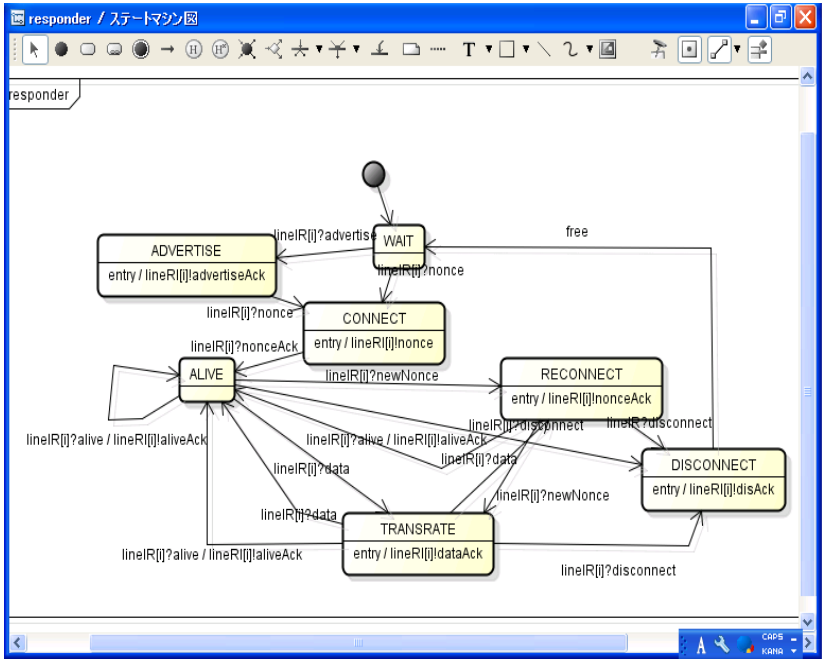

呉ヒョク，“P2P 仮想ネットワークにおける移動体接続の永続化プロトコル設計と検証”，信州大学大学院修士論文.

# 変換例

- ステートマシン図に，InitiatorとResponderの振る舞いを状態遷移で記述する



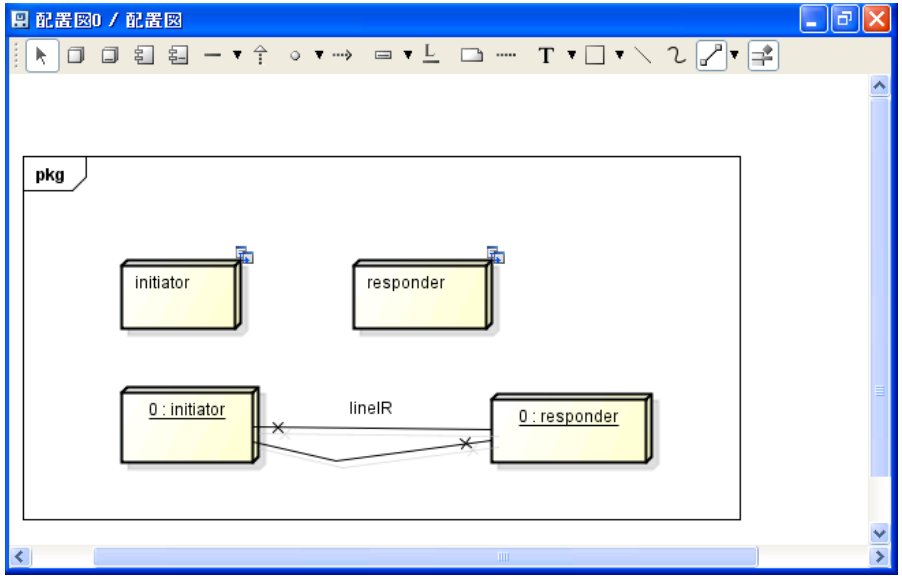

Initiatorのステートマシン図



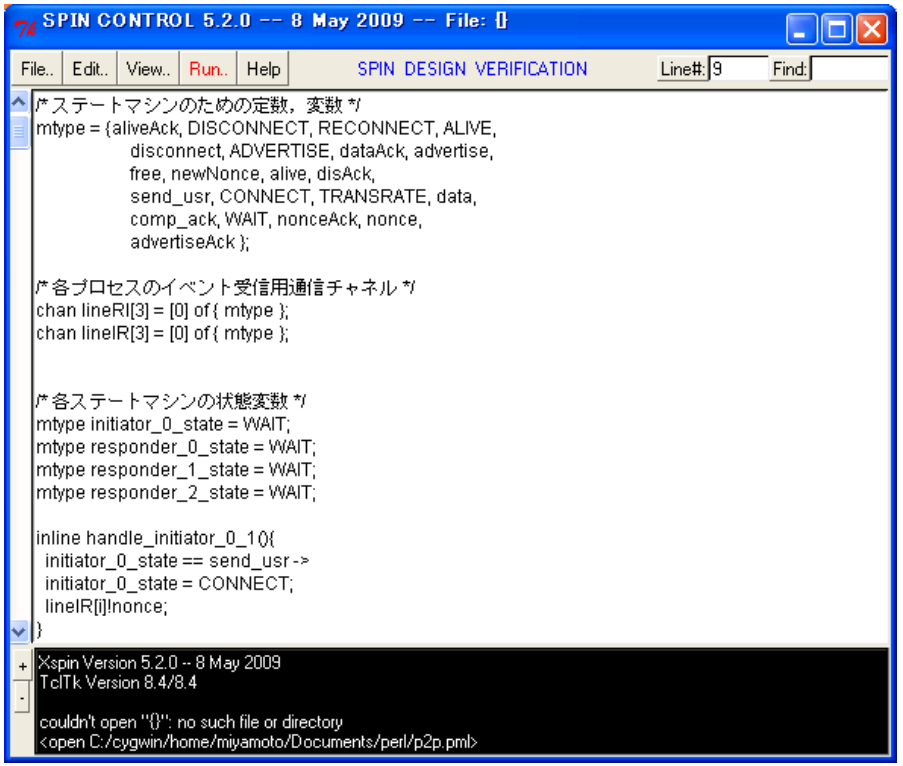

Responderのステートマシン図

# 変換例

- 配置図にInitiatorとResponderのインスタンスを記述し，それらをリンク線で繋ぐ
- 記述した配置図，ステートマシン図をPROMELAコードへ変換



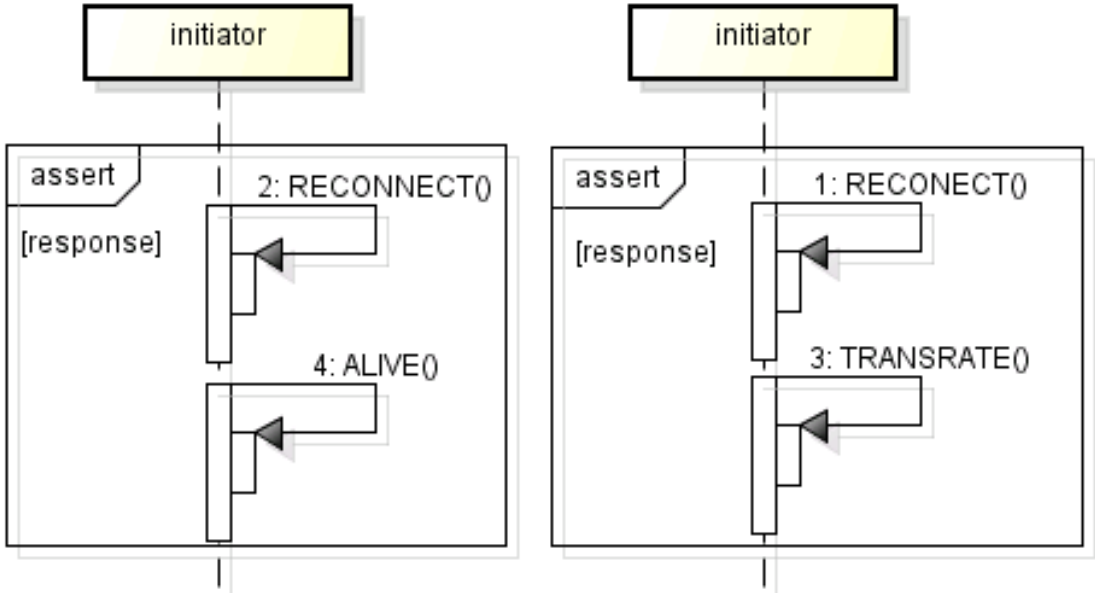

配置図




PROMELA

# 変換例

- シーケンス図で，要求仕様を記述
- 「initiatorはALIVE状態もしくはTRANSRATE状態に遷移したら，いつかは必ずRECONECT状態に遷移する」を検証する
- 応答性の検証のため，仕様パターンのresponseを記述する



# 変換例

- ステートマシン図，配置図をPROMELAコードに変換
- シーケンス図をLTL式に変換
- 変換されたコードと式で，SPINを用いてモデル検査した結果，要求仕様を満たすことを確認した

```
([](p_0 -> <>s_0)) || ([](p_1 -> <>s_1))
#define p_0 (initiator_0_state == RECONNECT)
#define p_1 (initiator_0_state == RECONNECT)
#define s_0 (initiator_0_state == ALIVE)
#define s_1 (initiator_0_state == TRANSRATE)
```

```
State-vector 40 byte, depth reached 173, errors: 0
      415 states, stored (485 visited)
      315 states, matched
      800 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.022 equivalent memory usage for states
                (stored*(State-vector + overhead))
    0.277 actual memory usage for states
                (unsuccessful compression: 1250.41%)
          state-vector as stored = 684 byte +
                                   16 byte overhead
    2.000 memory used for hash table (-w19)
    0.305 memory used for DFS stack (-m10000)
    2.501 total actual memory usage
```

# まとめ

- UML図で記述されたモデルから，PROMELAモデル及びLTL式に変換
  - ➔ 上流設計から，モデル検査までの一貫した設計検証環境を提供
- UMLを単一のツールで複数の図を統一して記述することにより，各図の整合性を取ることができる

# 今後の展望

- 自動変換の際，一意に変換するためUMLの記法を制限した
  - ➔ 他の記法を用い，より複雑なモデルの記述に対応させる
- PROMELAモデル変換器と，LTL式変換器を統合
  - ➔ より一貫した設計検証環境の実現
- モデル検査の結果，反例が生じた場合，UML記述ツールにフィードバック
  - ➔ UMLと照らし合わせたレビューを支援できる

# 参考文献


- [1] Gerard J.Holzmann, “THE SPIN MODEL CHECKER”,Addison-Wesley, 2004.
- [2] 吉岡信和青木利晃田原康之, “SPIN による設計モデル検証”, 近代科学社, 2009
- [3] 中島震, “SPIN モデル検査”, 近代科学社, 2008.
- [4] 宮本直樹, 和崎克己“UML 記述の仕様からSPIN モデル検査用 PROMELA でモデルへの自動変換”, 情報科学技術フォーラム, 2010.
- [5] 呉ヒョク, “P2P 仮想ネットワークにおける移動体接続の永続化プロトコル設計と検証”, 信州大学大学院修士論文.
- [6] 児玉公信, “UML モデリング入門”, 日経BP 社, 2008.
- [7] SPEC PATTERNS, http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml.
- [8] 株式会社チェンジビジョン, http://www.changevision.com/.