自動証明系と対話型証明支援系の連携によるポインタ操作プログラムの検証について

（MLATとAgdaの接合）

湯浅 能史（産総研） 武山 誠（産総研）
関澤 俊弦（産総研・阪大） 田辺 良則（産総研・東大）
高橋 孝一（産総研）

対話証明器 vs 自動証明器

- 対話型証明支援系
  長所：“あらゆる” 性質が 証明/反証可能。
  短所：人手による証明。手間がかかる。
- 自動定理証明器
  長所：機械任せ。“手間がかからない”。
  短所：示せる性質は限定的。

連携による検証が有効

Agda ＋ MLAT

- Agda ---- 対話型証明支援系。Martin-Löf型理論に基づく。Emacs による ユーザインターフェース。.
- MLAT ---- 抽象遷移系生成器。ポインタ操作プログラムが対象。様相論理を用いた述語抽象。

- 連携の方法 ---- Agdaプラグイン機構を利用。
- 検証の枠組み ---- Hoare 論理。
- 検証の対象 ---- ポインタ操作プログラム。手続き型。

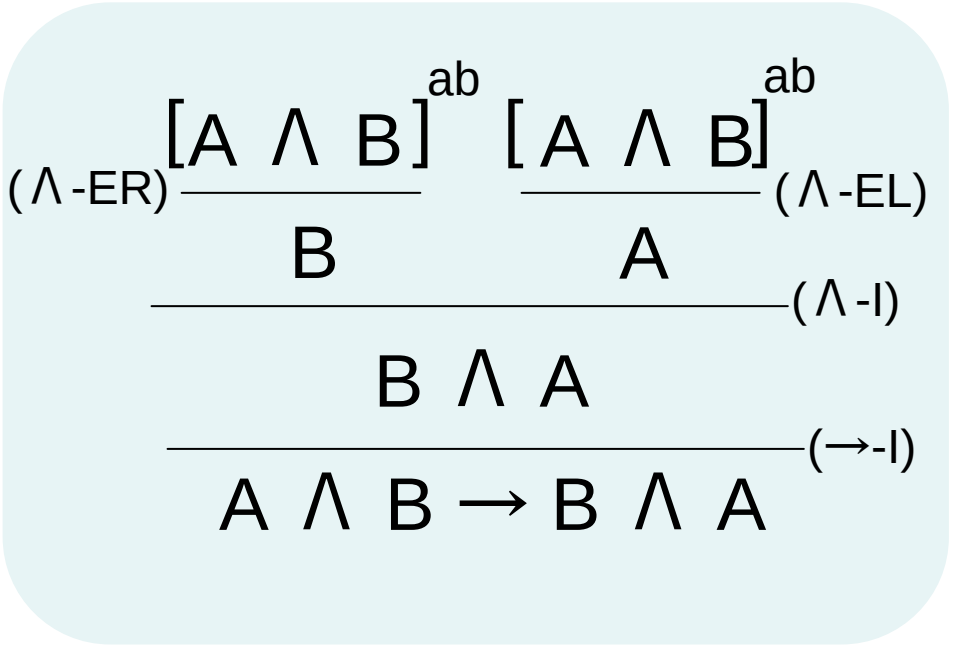
目次

- 対話証明器 Agda について
  - Agda による定理証明
  - Agda Prover Plug-in
- Pml-Hoare 論理
- 抽象遷移系生成器 MLAT について
  - MLAT の仕組み
  - ポインタ構造と時相論理式
- Agda-MLAT連携による検証例
  - リスト反転プログラムの検証
  - ループ不変式の確定
- まとめ

目次

- 対話証明器 Agda について
  - Agda による定理証明
  - Agda Prover Plug-in
- Pml-Hoare 論理
- 抽象遷移系生成器 MLAT について
  - MLAT の仕組み
  - ポインタ構造と時相論理式
- Agda-MLAT連携による検証例
  - リスト反転プログラムの検証
  - ループ不変式の確定
- まとめ

Agdaによる定理証明




```
comCnj (A,B :: Set) :: A ∧ B -> B ∧ A
    = ¥(ab :: A ∧ B)->
        and elimAndR ab
            elimAndL ab
```

Emacs

- Curry-Howard 対応に基づいた証明の符号化。
- 関数型プログラミング言語に似た証明記述。
- 対話的にゴールをサブゴールに展開する。

Agdaによる定理証明

ab :: A ∧ B

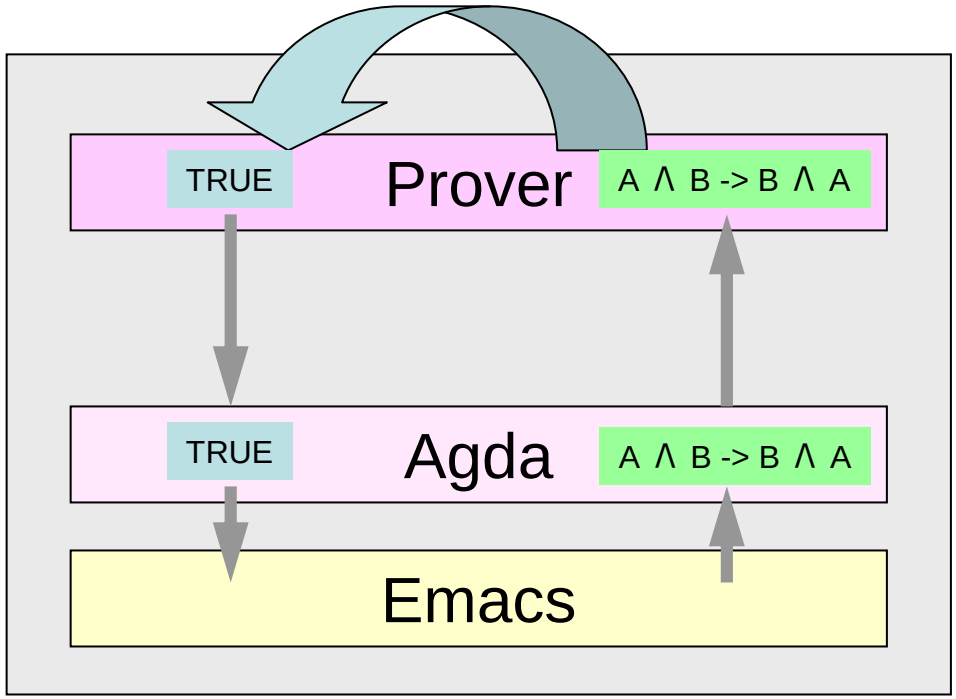
$$\dfrac{\dfrac{[A \wedge B]^{ab}}{B}(\wedge\text{-ER}) \quad \dfrac{[A \wedge B]^{ab}}{A}(\wedge\text{-EL})}{B \wedge A}(\wedge\text{-I})}{A \wedge B \rightarrow B \wedge A}(\rightarrow\text{-I})$$

```
comCnj (A,B :: Set) :: A ∧ B -> B ∧ A
    ≡ λ{ab!}: A ∧ B)->{!   !}
        and{!ab!}
            {!and!}
```

Emacs

- Curry-Howard 対応に基づいた証明の符号化。
- 関数型プログラミング言語に似た証明記述。
- 対話的にゴールをサブゴールに展開する。

Agda Prover Plug-in




```
comCnj (A,B :: Set) :: A ∧ B -> B ∧ A
    = {external "FOL"!}
```

Emacs

- 既存の自動証明器に手を加えることなく、Agda側の設定でだけで接続できる。
- これまで接続した自動証明器:
  - gandalf (FOL)
  - NuSMV

目次

- 対話証明器 Agda について
  - Agda による定理証明
  - Agda Prover Plug-in
- Pml-Hoare 論理
- 抽象遷移系生成器 MLAT について
  - MLAT の仕組み
  - ポインタ構造と時相論理式
- Agda-MLAT連携による検証例
  - リスト反転プログラムの検証
  - ループ不変式の確定
- まとめ

PML-Hoare 論理

Pointer Manipulation Language

- 基本条件式 AS ： x == Null, x
- 基本命令文 AC ： x := Null, x.
  x := y.f , x.
- プログラム

条件式： S ::= AS | (! S) | (S && S) | (S || S)
else {CS} | while S do {CS}

Hoare Triple（Hoare 式）

P {cs} Q

cs 実行前に P なら 実行後には Q

最弱前条件：
・命令 c 実行前に wp_c Q ならば実行後にはQ

Agda上に”深い符号化”で実現

Pformula（後述）

$$\frac{P \Rightarrow wp_c\, Q}{P\,\{c\}\,Q}\ (\mathrm{atm})\quad (\text{for } c \in AC)$$

$$\frac{P \wedge s\,\{cs_0\}\,Q \quad P \wedge \neg s\,\{cs_1\}\,Q}{P\,\{\mathtt{if}\ s\ \mathtt{then}\ \{cs_0\}\ \mathtt{else}\ \{cs_1\}\}\,Q}\ (\mathrm{ite})$$

MLATで自動証明

$$\frac{P \Rightarrow R \quad R \wedge s\,\{cs\}\,R \quad R \wedge \neg s \Rightarrow Q}{P\,\{\mathtt{while}\ s\ \mathtt{do}\ \{cs\}\}\,Q}\ (\mathrm{whl})$$

$$\frac{P \Rightarrow Q}{P\,\{\lambda\}\,Q}\ (\mathrm{nil})$$

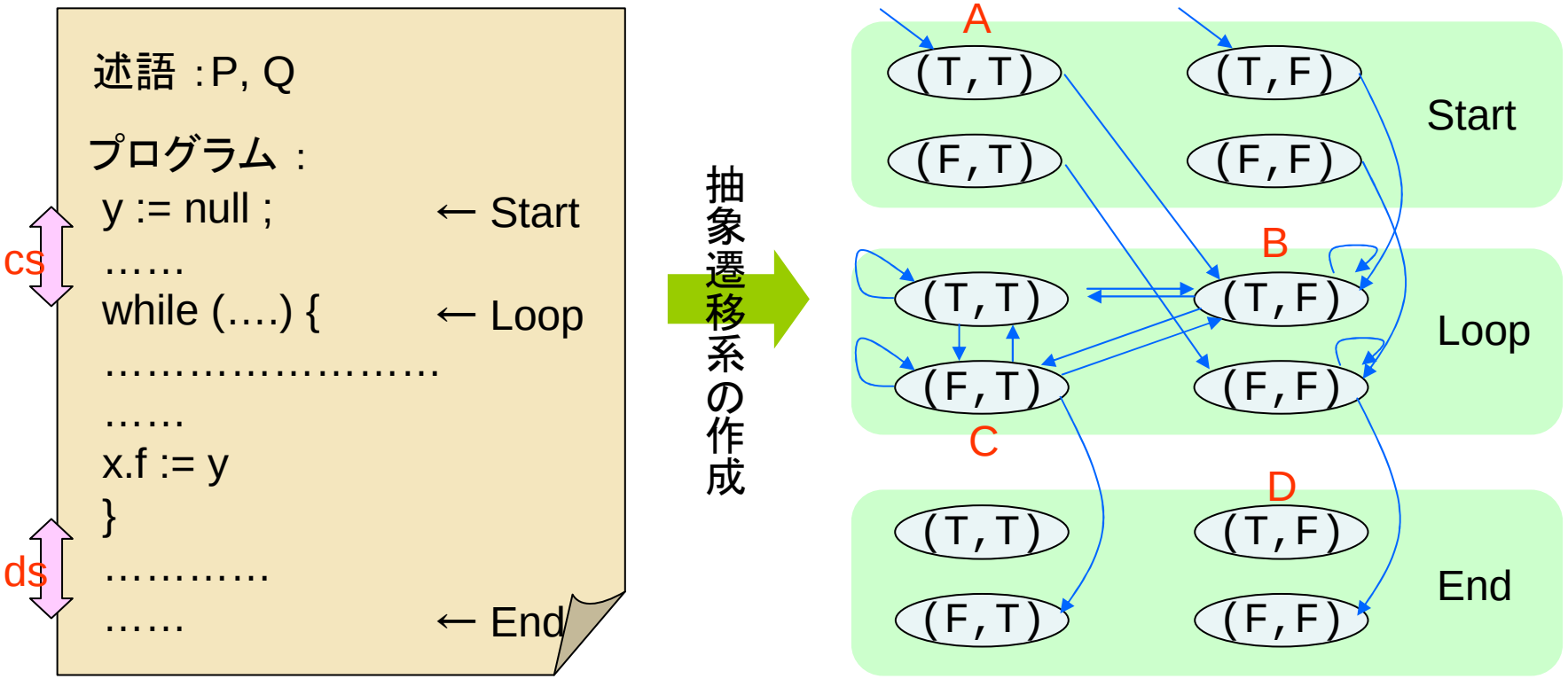
$$\frac{P\,\{c\}\,R \quad R\,\{cs\}\,Q}{P\,\{c;cs\}\,Q}\ (\mathrm{cns})$$

（制限付き）

目次

- 対話証明器 Agda について
  - Agda による定理証明
  - Agda Prover Plug-in
- Pml-Hoare 論理
- 抽象遷移系生成器 MLAT について
  - MLAT の仕組み
  - ポインタ構造と時相論理式
- Agda-MLAT連携による検証例
  - リスト反転プログラムの検証
  - ループ不変式の確定
- まとめ

MLAT (Modal Logic Abstraction Tool)



「最弱前条件(wp)」と「充足可能性判定(sat)」により遷移可能性を判定する。

sat(A ∧ wp(cs, B)) = true ⇒ 遷移あり

sat(C ∧ wp(ds, D)) = false ⇒ 遷移なし

pfvalid(X,cs,Y) =
not sat(X ∧ wp(cs, not Y))

Hoare式の判定

ポインタ構造と時相論理

プログラム実行のある時点でのヒープの状態をリンク構造とみなす

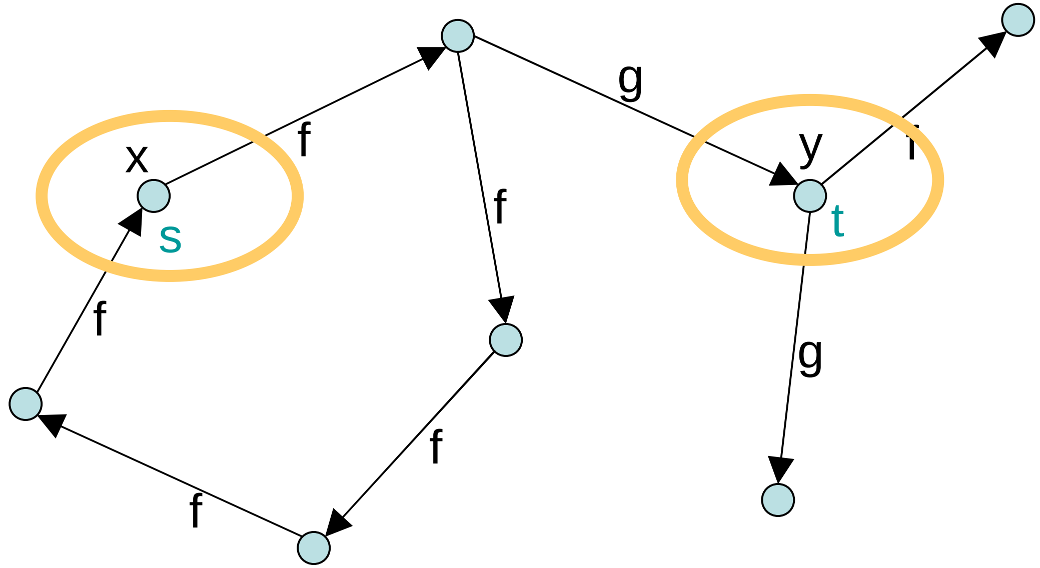
性質を様相CTL式で記述する。

変数 :x, y,....

フィールド :f, g,....

プログラム:

```
y := null ;
……
while (….) {
……………………
……
x.f := y
}
…………
……
```



変数 x がノード s をさす: s ⊨ x　　$E_f X$ $E_f X$ $E_g X$ y

変数 y がノード t をさす. (s ⊨ x → s ⊨ $E_f X E_g X$ y))

PML-Hoare 論理

- 基本条件式 AS ： $x == Null,\quad x.val == b,\quad x == y$
- 基本命令文 AC ： $x := Null,\quad x.val := b,\quad x := y,$ $x := y.f,\quad x.f := y,\quad x := new$
- プログラム

  条件式： S ::= AS | (S && S) | (S || S)

  命令文： C ::= AC | if S then {CS} else {CS} | while S do {CS}

  プログラム： CS ::= λ | C;CS

Pformula:
式 “x |- P” の命　　合（x: プログラム変数，P: CTL式）

例：(x |- P) ⇒ (y |- Q) ∧ (z |- R)

Agda上に”深い符号化”で実現

- 推論規則 （P,Q,R：ヒープの状態を記述する論理式

$$\frac{P \Rightarrow \mathrm{wp}_c\, Q}{P\,\{c\}\,Q}\ (\mathrm{atm})\quad (\text{for } c \in \mathrm{AC})$$

MLATで自動証明

$$\frac{P \wedge \bar{s}\,\{cs_0\}\,Q \quad P \wedge \neg\bar{s}\,\{cs_1\}\,Q}{P\,\{\mathtt{if}\ s\ \mathtt{then}\ \{cs_0\}\ \mathtt{else}\ \{cs_1\}\}\,Q}\ (\mathrm{ite})$$

$$\frac{P \Rightarrow R \quad R \wedge \bar{s}\,\{cs\}\,R \quad R \wedge \neg\bar{s} \Rightarrow Q}{P\,\{\mathtt{while}\ s\ \mathtt{do}\ \{cs\}\}\,Q}\ (\mathrm{whl})$$

$$\frac{P \Rightarrow Q}{P\,\{\lambda\}\,Q}\ (\mathrm{nil})$$

$$\frac{P\,\{c\}\,R \quad R\,\{cs\}\,Q}{P\,\{c;cs\}\,Q}\ (\mathrm{cns})$$

（制限付き）

目次

- 対話証明器 Agda について
  - Agda による定理証明
  - Agda Prover Plug-in
- Pml-Hoare 論理
- 抽象遷移系生成器 MLAT について
  - MLAT の仕組み
  - ポインタ構造と時相論理式
- Agda-MLAT連携による検証例
  - リスト反転プログラムの検証
  - ループ不変式の確定
- まとめ

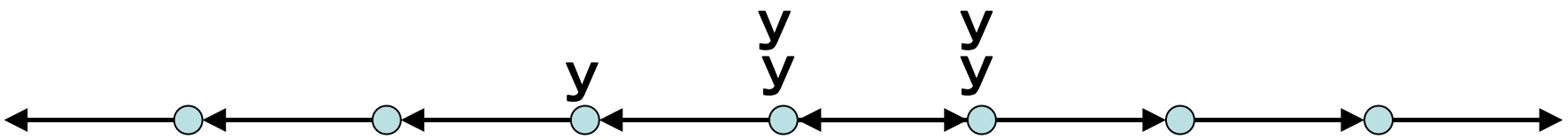
検証例「リスト反転プログラム」

reverse :

```
y := Null ;
while (not (x == Null)) do {
  t       := y ;
  y       := x ;
  x       := x.next ;
  y.next  := t ;
} ;
```

ループを一　すると :

- t, y, x が　に移動
- 　　が一つ反転



Agdaで

Listp x := (x |- EF NULL)

x から NULL (= nil)に する

前/ 後条件

```
Pre :: PFormula = ListP x && Reach x u && Next u v && not (Nullp v)
Pos :: PFormula = Next v u
```

ループ不変式

```
Inv ::
```

A

$$\frac{\dfrac{\mathrm{Inv} \Rightarrow \mathrm{Inv} \quad \mathrm{Inv} \wedge \neg\mathrm{Nullp}\, x \;\{\ldots\}\; \mathrm{Inv} \quad \mathrm{Inv} \wedge \neg\neg\mathrm{Nullp}\, x \Rightarrow \mathrm{Pos}}{\mathrm{Inv}\;\{\texttt{while}\,(\texttt{x} \neq \texttt{Null})\,\texttt{do}\,\{\ldots\}\}\;\mathrm{Pos}} \qquad \mathrm{Pos} \Rightarrow \mathrm{Pos}}{\mathrm{Pre}\;\{\texttt{y} := \texttt{Null}\}\;\mathrm{Inv} \qquad \dfrac{\mathrm{Inv}\;\{[\texttt{while}\,(\texttt{x} \neq \texttt{Null})\,\texttt{do}\,\{\ldots\}]\}\;\mathrm{Pos}}{\mathrm{Pre}\;\{\texttt{reverse}\}\;\mathrm{Pos}}}$$

Hoare

```
rev :: HT Pre reverse Pos
    = let lem1 :: HTc Pre (setNull y) Inv  = external "mlat"
          lem2 :: HTp (Inv && not (Nullp x)) loopBody Inv = external "mlat"
          lem3 :: PFImp (Inv && not (not (Nullp x))) Pos  = external "mlat"
      in cns_ lem1 (cns_ (whl_ id_ lem2 lem3) (nil_ id_))
```

MLATによる自動証明

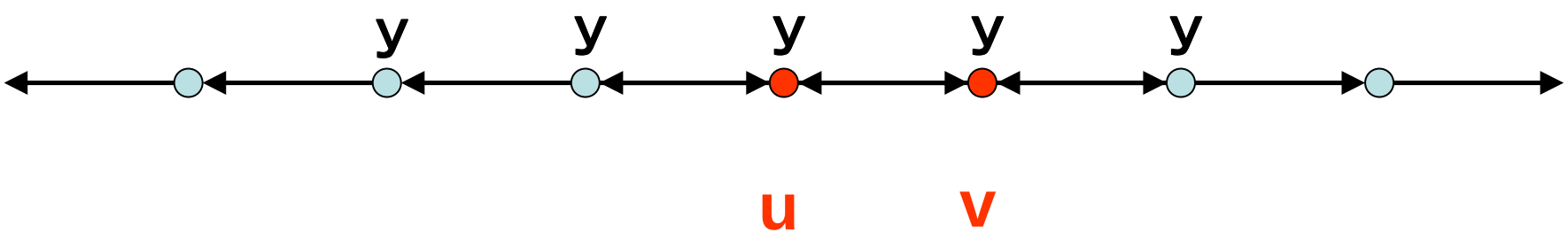
ループ不変式の確定

1st trial

```
ListP x && not (Nullp v) &&
    (     Reach x u  && Next u v ||
                        Equal x v && Equal y u ||
                        Next v u)
```

ループ不変式

→ “No”（MLAT）



ループ不変式の確定

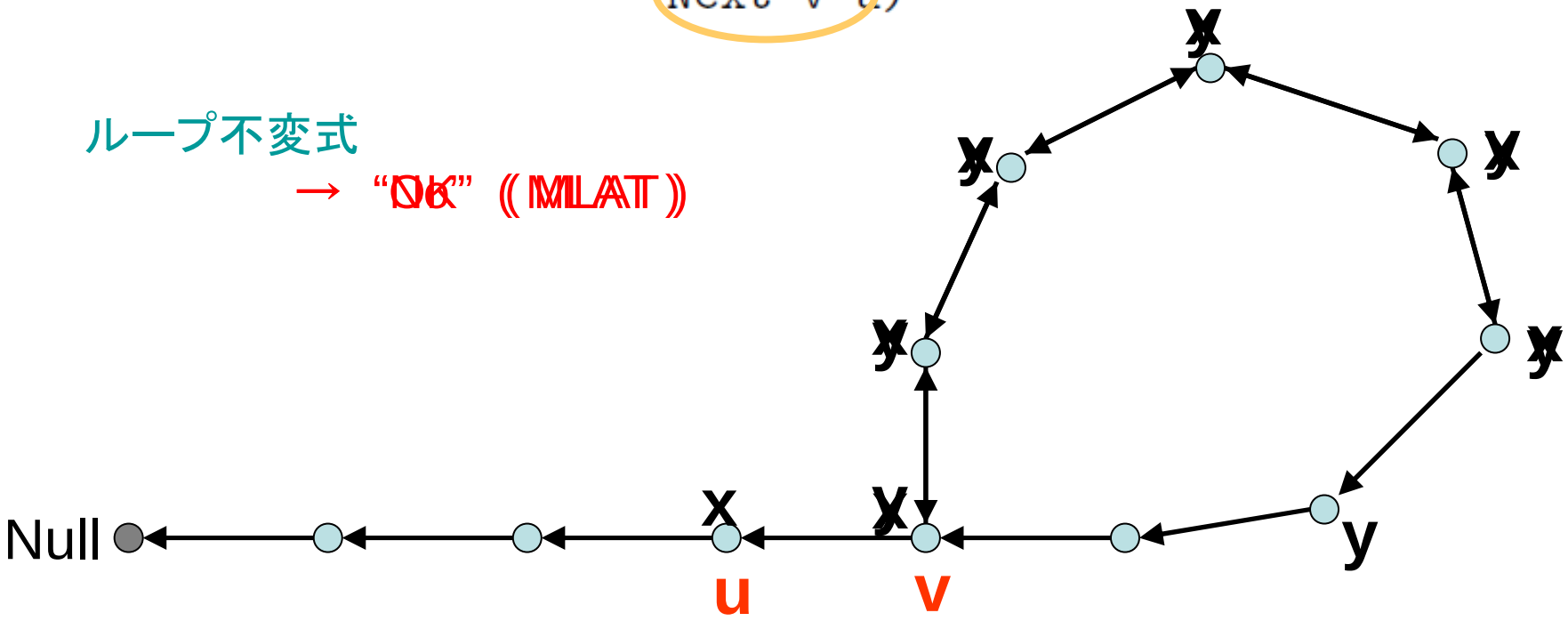
3rd trial

```
ListP x && not (Nullp v) &&
    (     Reach x u  && Next u v ||
                        Equal x v && Equal y u ||
                        Next v u)
```

?

ループ不変式

→ “OK” (MLAT)



Agdaでの証明記述

```
Pre :: PFormula = ListP x && Reach x u && Next u v && not (Nullp v)
Pos :: PFormula = Next v u

Inv :: PFormula = ListP x && not (Nullp v) &&
      (Reach x u  && Next u v ||
       not (Reach x u) && Equal x v && Equal y u ||
       not (Reach x u) && Next v u)

rev :: HT Pre reverse Pos
    = let lem1 :: HTc Pre (setNull y) Inv  = external "mlat"
          lem2 :: HTp (Inv && not (Nullp x)) loopBody Inv = external "mlat"
          lem3 :: PFImp (Inv && not (not (Nullp x))) Pos  = external "mlat"
      in cns_ lem1 (cns_ (whl_ id_ lem2 lem3) (nil_ id_))
```

ループ不変式

Agda-MLAT連携による 行 で

MLATによる自動証明

目次

- 対話証明器 Agda について
  - Agda による定理証明
  - Agda Prover Plug-in
- Pml-Hoare 論理
- 抽象遷移系生成器 MLAT について
  - MLAT の仕組み
  - ポインタ構造と時相論理式
- Agda-MLAT連携による検証例
  - リスト反転プログラムの検証
  - ループ不変式の確定
- まとめ

まとめ:　　と　後の

自動-対話　　の　リ　ト

- 対話証明器を用いた　　により、自動証明にかかる時間を短　できる。（　0s →10s）
- 　行　　の　　として自動証明器を用いることができる。(ループ不変式の　　な　）

後の

- 最弱　前条件の　　。(現状では、Hoare式の証明は　MLATでしか　できない）
- MLATからAgda　の　　の　け　し。(　のときに反例、　のときに証明の　ヒント）
- 大きな例　、　　の　　、etc.

り