講義 3: エージェント指向プログラミング

Agent Oriented Programming

内容

1. AGENT0
   - 能力，信念，コミット メント
   - 伝達行為 (communicative act)
2. 並行 MetateM
   - 様相演算子
   - 時相論理プログラミング
3. 参考文献

1. AGENT0(1/10)

- AI 研究コミュニティのエージェントに対する興味の多くは Shoham のエージェント指向プログラミング (AOP)[Shoham90,Shoham93] の概念から生じてきた.
- 新計算パラダイム AOP は計算の社会的考察に基づく.
- AOP の重要なアイデアは，エージェントの特性を表現するために心的概念 (信念，願望，意図など) により，エージェントを直接プログラミングすること.
- この提案の背後にある動機は，人間は複雑なシステムの特性を表現する抽象化メカニズムとして，そのような概念を使うということ.

$\Longrightarrow$ 人間の振舞いの記述や説明に心的概念を使うのと同じように，機械をプログラムするのにそれらを使用したら有用であろう!!

でも，なぜそう言えるのか?

1. AGENT0(2/10)

- 完全な AOP システムは 3 つの構成要素を持つ [Shoham].
  - エージェントの心的状態を規定し記述する論理.
  - エージェントをプログラミングするインタプリタ型プログラミング言語.
  - 自然なアプリケーションをエージェントに変換するためのエージェント化 (agentification) プロセス.

  研究論文では最初の二つの構成要素について報告している.

- ここでは論理をスキップして AOP 言語について考える.

1. AGENT0(3/10)

- AGENT0 は LISP の拡張として実装された．AGENT0 のエージェントは４つの構成要素を持つ．
  - 能力 (エージェントが出来ること) の集合
  - 初期信念の集合
  - 初期コミット メント の集合
  - コミット メント 規則の集合
- 鍵となる構成要素はコミット メント 規則の集合

1. AGENT0(4/10)

- 各コミット メント 規則は以下を含む.
  - メッセージ条件
  - 心的条件
  - 行為

- 各エージェント サイクルで，規則が発火するかどうかを決定するために，

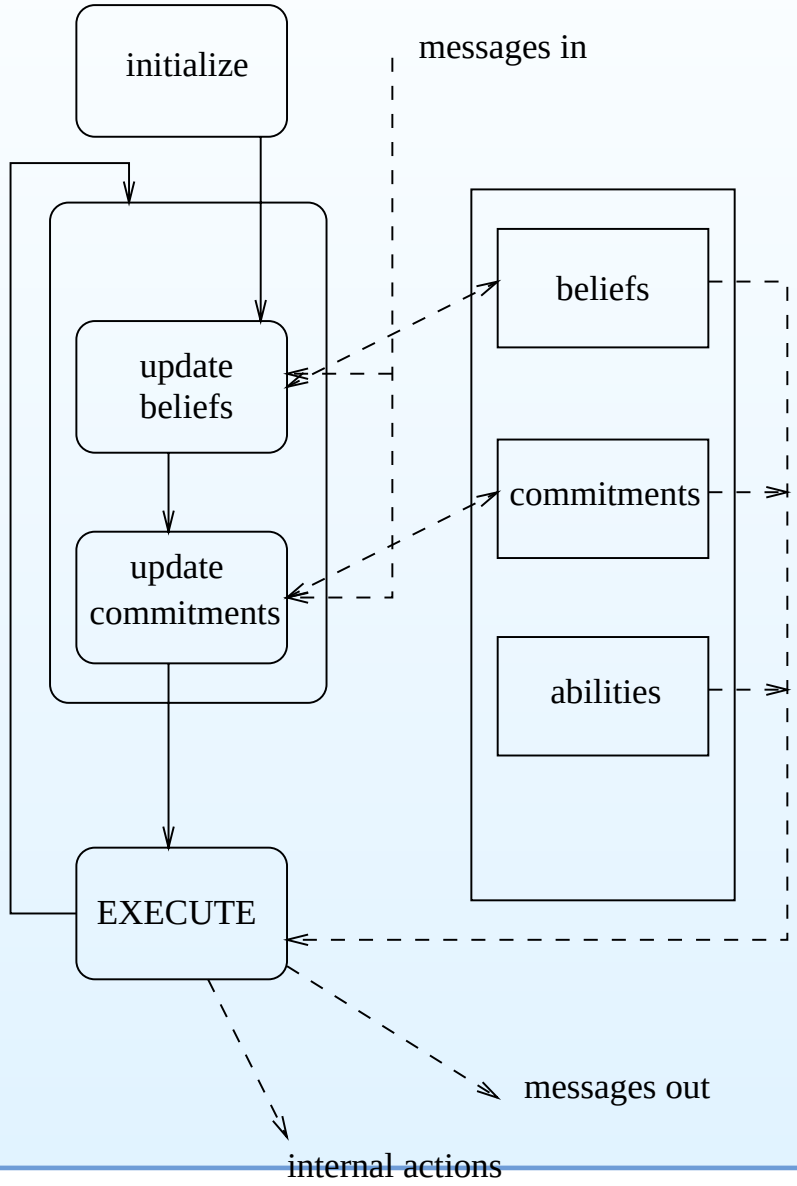
1. エージェントが受信したメッセージに対してメッセージ条件が照合される
2. エージェント の信念に対して心的条件が照合される
3. もしも規則が発火すると，エージェントはその行為にコミットする

1. AGENT0(5/10)

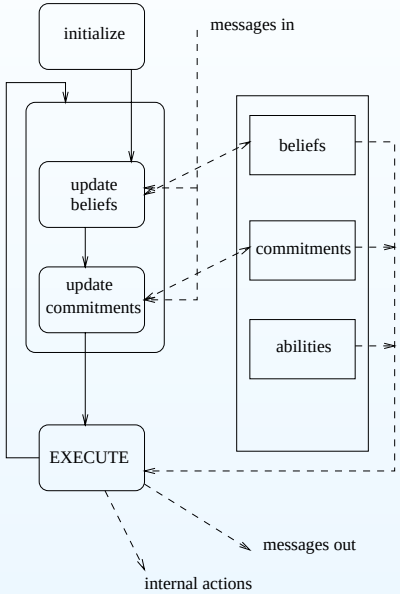
- 行為は以下の 2 種類
  - 内部的 (private): 内部実行されるサブルーチンに対応するもの
  - 伝達的 (communicative): メッセージ送信
- メッセージは以下の 3 つのタイプのうちの一つに制限される

1. 要求 (request): 行為へのコミット メントを要求する
2. 非要求 (unrequest): 行為を行わないよう非要求する
3. 伝達 (inform): 情報を伝える

1. AGENT0(6/10)


initialize
messages in
update beliefs
update commitments
EXECUTE
beliefs
commitments
abilities
messages out
internal actions

1. AGENT0(7/10)



1. すべての現在のメッセージを読み，必要な場合には信念，そしてそれによりコミット メントを，更新する.
2. 関連づけられた行為の能力条件が満たされたら，現在のサイクルに対するすべてのコミット メントを実行する.
3. (1) へ行く.

1. AGENT0(8/10)

- コミット メント 規則の例.

```
COMMIT(
  ( agent, REQUEST, DO(time, action)
  ), ;;; msg condition
  ( B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyaction)]
  ), ;;; mental condition
  self,
  DO(time, action)  )
```

1. AGENT0(9/10)

- この規則は次のように言い替えられる

  もし私が $time$ において $action$ を行うことを要求するメッセージを $agent$ から受信し，私は以下のことを信じていたら，

  - $agent$ は現在，友人である;
  - 私はその行為を実行できる;
  - $time$ において，如何なる行為にもコミットしていない，

  $time$ において $action$ を行うことにコミットする．

1. AGENT0(10/10)

- AGENT0 はマルチエージェントにおける協調と対話を支援する.
- しかし，AGENT0 はある原理を例証するためのプロトタイプであり，生産言語ではない.
- より洗練された PLACA という実装がある [Thomas1993]

2. 並行 MetateM(1/12)

並行 MetateM 言語 [Fischer94] と時相論理

- エージェントの挙動を時相論理で仕様記述してプログラミングするマルチエージェント言語
- 仕様は直接実行されエージェントの挙動を生成
- 時相論理は様相演算子により拡張された古典論理 (命題の真理値が時間とともにどのように変化するかを記述できる)

2. 並行 MetateM(2/12)

様相演算子の例

Always : $\Box \mathsf{Important}(\mathsf{agents}) \stackrel{\mathsf{def}}{=}$ “it is now, and will always be true that agents are important”

Sometime in the future : $\Diamond \mathsf{Important}(\mathsf{ConcurrentMetateM}) \stackrel{\mathsf{def}}{=}$ “sometime in the future, ConcurrentMetateM will be important”

Sometime in the past : $\blacklozenge \mathsf{Important}(\mathsf{Prolog}) \stackrel{\mathsf{def}}{=}$ “sometime in the past, Prolog was important”

Until : $(\neg \mathsf{frined}(\mathsf{us}))\, \mathcal{U}\, \mathsf{apologise}(\mathsf{you}) \stackrel{\mathsf{def}}{=}$ “we are not frineds until you apologise”

Next : $\odot \mathsf{apologise}(\mathsf{you}) \stackrel{\mathsf{def}}{=}$ “tomorrow (in the next state),apologise”

2. 並行 MetateM(3/12)

様相演算子一覧

Operator	Meaning
$\odot\varphi$	φ is true ‘tomorrow’
$\bigcirc\varphi$	φ was true ‘yesterday’
$\Diamond\varphi$	at some time in the future, φ
$\Box\varphi$	always in the future, φ
$\blacklozenge\varphi$	at some time in the past, φ
$\blacksquare\varphi$	always in the past, φ
$\varphi\,\mathcal{U}\,\psi$	φ will be true until ψ
$\varphi\,\mathcal{S}\,\psi$	φ has been true since ψ
$\varphi\,\mathcal{W}\,\psi$	φ is true unless ψ
$\varphi\,\mathcal{Z}\,\psi$	φ is true zince ψ

2. 並行 MetateM(4/12)

- MetateM のルーツは Gabbay の分離論理
  任意の時相論理式は，論理的に等価な $past \Rightarrow future$ 形式に書き換えることができる
- $past \Rightarrow future$ 形式は実行規則として使える
- MetateM のプログラムはそのような規則の集合
- 実行は，継続的に規則を履歴と照合し，前提条件が充足された規則を見つける
- 具現化された未来の結論部はその後に充足されるコミットメントとなる

2. 並行 MetateM(5/12)

- よって，実行はプログラム規則の論理式に対するモデルを繰り返し生成する過程となる
- 具現化された規則の未来時間部分はこのモデルの制約を表現する
- MetateM プログラムの例: 資源制御

  $\forall x \; \bigcirc \, ask(x) \Rightarrow \Diamond give(x)$

  $\forall x, y \;\; give(x) \wedge give(y) \Rightarrow (x = y)$

- 最初の規則は ‘ask’ のあとにいつか ‘give’ がくることを保証する
- 二番目の規則は，一時にはただ一つの ‘give’ だけが遂行されることを保証する

2. 並行 MetateM(6/12)

- 並行 MetateM は MetateM のプロセスの集まりが作用し通信することのできる操作的枠組を提供する
- それは実行可能論理の新しい並行モデルに基づいている．それは，個々のエージェントの挙動を生成するために論理仕様を実行するという概念
- 並行 MetateM システムは多数のエージェントを含み，各エージェントは 3 つの属性を持つ

  1. 名前
  2. インタフェース
  3. MetateM プログラム

2. 並行 MetateM(7/12)

- エージェントのインタフェースは二つの集合を持つ
  - 環境述語: エージェントが受理するメッセージに相当
  - 構成要素述語: エージェントが送信するメッセージに相当
- 事例: 'stack' エージェントのインタフェース

$$stack(pop, push)[popped, stackfull]$$

$\{pop, push\} =$ 環境述語
$\{popped, stackfull\} =$ 構成要素述語

- 環境述語を頭部にもつメッセージをエージェントが受信すると，それは受理される
- エージェントが，構成要素述語に相当するコミットメントを充足できれば，それをブロードキャストする．

2. 並行 MetateM(8/12)

並行 MetateM のプログラム事例 (白雪姫)

- 白雪姫はお菓子 (資源) を持っていて，それをコビト (資源消費者) にあげることになる
- 彼女は同時には一人のコビトにだけお菓子あげる
- 彼女はいつも，お菓子を欲しがるコビトにはいつかそれをあげる
- これを並行 MetateM で記述すると以下のようになる.

$$
\begin{array}{l}
SnowWhite(ask)[give]: \\
\qquad\qquad \textcircled{\scriptsize O}\, ask(x) \;\Rightarrow\; \Diamond give(x) \\
\qquad give(x) \wedge give(y) \;\Rightarrow\; (x = y)
\end{array}
$$

2. 並行 MetateM(9/12)

- コビト ‘eager’ が最初にお菓子を欲しがり，そしてお菓子を受け取ると，また欲しがる．

$$eager(give)[ask]:$$
$$\text{start} \Rightarrow ask(eager)$$
$$\bigcirc give(eager) \Rightarrow ask(eager)$$

- あるコビト ‘greedy’ は礼儀正しくなくて，いつもお菓子を欲しがる．

$$greedy(give)[ask]:$$
$$\text{start} \Rightarrow \Box ask(greedy)$$

2. 並行 MetateM(10/12)

- 運良く，あるコビトはマナーがよい．‘courteous’ は ‘eager’ と ‘greedy’ が食べ終った時にお菓子を欲しがる

$$courteous(give)[ask]:$$
$$((\neg ask(courteous)\,\mathcal{S}\,give(eager))\wedge$$
$$(\neg ask(courteous)\,\mathcal{S}\,give(greedy))) \quad \Rightarrow \quad ask(courteous)$$

- 最後に，‘shy’ は他に誰も欲しがっていないときにだけお菓子を欲しがる

$$shy(give)[ask]:$$
$$\text{start} \quad \Rightarrow \quad \Diamond ask(shy)$$
$$\bigcirc ask(x) \quad \Rightarrow \quad \neg ask(shy)$$
$$\bigcirc give(shy) \quad \Rightarrow \quad \Diamond ask(shy)$$

2. 並行 MetateM(11/12)

$$
\begin{array}{rcl}
SnowWhite(ask)[give]: & & \\
\bigcirc ask(x) & \Rightarrow & \Diamond give(x) \\
give(x) \wedge give(y) & \Rightarrow & (x = y) \\
 & & \\
eager(give)[ask]: & & \\
\text{start} & \Rightarrow & ask(eager) \\
\bigcirc give(eager) & \Rightarrow & ask(eager) \\
 & & \\
greedy(give)[ask]: & & \\
\text{start} & \Rightarrow & \Box ask(greedy) \\
 & & \\
courteous(give)[ask]: & & \\
((\neg ask(courteous)\, \mathcal{S}\, give(eager)) \wedge & & \\
(\neg ask(courteous)\, \mathcal{S}\, give(greedy))) & \Rightarrow & ask(courteous) \\
 & & \\
shy(give)[ask]: & & \\
\text{start} & \Rightarrow & \Diamond ask(shy) \\
\bigcirc ask(x) & \Rightarrow & \neg ask(shy) \\
\bigcirc give(shy) & \Rightarrow & \Diamond ask(shy)
\end{array}
$$

2. 並行 MetateM(12/12)

まとめ

- 実験的言語
- きれいな基礎論理
- 現在はプロトタイプのみで，完全実装はまだ作業の途中

参考文献 (1/2)

Shoham90 Shoham, Y. (1990) Agent-oriented programming. Technical report STAN-CS-1335-90, Computer Science Department, Stanford University, Stanford, CA 94305.

Shoham93 Shoham, Y. (1993) Agent-oriented programming. *Artificial Intelligence*, **60**(1), 51–92.

Fischer94 Fisher, M. (1994) A survey of Concurrent MetateM–the language and its applications. In *Temporal Logic – Proceedings of the 1st International Conference* (eds D. M. Gabbay and H. J. Ohlbach), LNAI Volume 827, pp. 480–505. Springer, Berlin.