OCC theoryによる感情表現を持つエージェントの実現

奈良女子大学　理学部　情報科学科4回生 10251781
新出研究室　山根瑞樹

概要

近年、ロボットの進化が著しい中で、人間との対話が重要視されている。それに伴い感情表現の付加が必要とされている。従来、OCC theory と呼ばれる、感情をモデル化する理論に基づき感情を生起し、自力でそれに伴った行動を選択できる BDI エージェントの実現を目指した研究があるが、全ての感情を汎用的に利用可能な形で実装するには至っていなかった。

そこで本研究では、今まで実装されてきた感情表現の定義を統一し、実装出来ていなかった感情表現を実装、それらの検証と今後の問題点の議論を行った。

1 はじめに

近年、ロボットの進化が著しい中で、人間との対話が重要視されている。それに伴い感情表現の付加が必要とされている。特に人とコミュニケーションをとるエージェントは実世界において感情を持ち行動出来ることが望まれる。

実世界のような動的環境において問題解決の為に行動できるエージェントの実現には、BDI アーキテクチャが有効であると示されている [8]。BDI アーキテクチャとは、BDI モデル [7] による行為決定方式を計算機上で実現したもので、これにより BDI エージェントという人間の合理的な思考に基づいた行動を取るエージェントを構築する事が可能である。BDI モデルとは、信念 (Belief)、願望 (Desire)、意図 (Intention) の 3 つの心的パラメータを用いて人間の合理的な行動をモデル化したものであり、BDI logic という論理モデルを持つ。

一方、OCC theory[6] と呼ばれる、心理学的見地に基に人間の包括的な感情を 22 種類のタイプに分類しモデル化した理論が感情の形式化に多く用いられている。この OCC theory は信念や願望などの心的状態を用いて感情の特徴付けが明確であり、また、その感情の特徴付けを論理モデルで表現可能という特徴を持つため、BDI logic を持つ BDI モデルとは親和性が高く、既存の BDI アーキテクチャの実装である Jason[5] 上で信念ベースを用いて実現することが可能である。

BDI モデルと OCC theory を用いてエージェントの感情を扱っている関連研究としては [1, 2] の他に [3] や [4] がある。

Adam ら [3] は、BDI モデルでの形式化に用いられる論理体系である BDI logic に対し、「不確実だが起こると期待されている事柄」を表現するなどいくらかの新たなオペレータを導入して拡張し、これを用いて、OCC theory で扱われる 20 種類の感情の特徴付けを論理式として形式化することで BDI モデルに取り込んでいる。

ただし、この研究におけるモデルでは一つの行動が一つの感情を生起されるようになっており、複合的な感情の生起については扱えない。また、感情の度合いについても扱っていない。加えて、現段階の実装としては、状況からの感情の生起を推論する部分を独自に実現したものはあるが、他のシステムと結合できるよう汎用的にまとめられたものはなく、感情表現をエージェントの行動決定の一部として取りこめるようにはなっていないため、BDI エージェント部分の実装を一般利用者が行えない。

この他、[4] では OCC theory を用いた BDI エージェントによって、学生の勉学を奨励させる教育用エージェントシステムを実現している。これは OCC theory を基にした推論システムによって生徒の感情を推論することで、適切な指導を行うことを目的としたもの

であり、生徒の勉学に対する姿勢を 2 つに大別した上で、それを基に全ての生徒に起こり得る状態を列挙し、感情の流れを記述している。しかし、人間の感情の推論を目指したものであり、エージェントやロボットの感情の生起を扱うものではない。

上記の BDI エージェントと OCC theory を用いて、感情を生起し自力でその感情に伴った行動が選択できるエージェントの Jason 上での実現を目指した研究 [1, 2] があるが、全ての感情を実現するには至っていない。特にそれぞれの感情が別の形式で定義されており、2 つの感情の生起が条件となるような感情が実現できていないという問題があった。

そこで本研究では、過去の研究で実現された感情の定義の形式を統一し、未実装であった感情も実装することで OCC theory にて定義されている 22 種類の感情の内、述語論理を要するため [3] で扱われていない 2 種類の感情を除く 20 種類の感情表現を実装した。

2 OCC theory

OCC theory[6] とは、Ortony, Clore, Collins らが提唱した、心理学的見地を基にした 22 種類の感情のタイプをモデル化した理論である。人間の包括的な感情を形式化しており、感情の特徴付けが明確であるため、比較的理解しやすく、計算機科学の分野において広く用いられている。

22 種類の感情をまとめると以下のようになる。

1. 事象の結果の望ましさにのみ焦点を当てたもの
   - (a) 結果の望ましさに関して (自分にとって)
     - i. 起こった事象に関するもの
       — Joy(喜び), Distress(嘆き)
     - ii. 予想に関するもの
       - A. 単なる予想に対して
         — Hope(望み), Fear(恐れ)
       - B. 予想していたことが起こったことに関して
         — Satisfaction(満足), FearConfirmed(恐れていたことが確定)
       - C. 予想していたことが起こらなかったことに関して
         — Relief(安堵), Disappointment(落胆)
   - (b) 結果の望ましさに関して (他者にとって)
     - i. 他者が良い結果を得た場合
       — HappyFor(共に喜ばしく思う), Resentment(憤り)
     - ii. 他者が悪い結果を得た場合
       — SorryFor(共に残念に思う), Gloating(ほくそ笑む)
2. 行動に対する賞賛度にのみ焦点を当てたもの
   - (a) 自分の行動に関するもの
     — Pride(自尊心), Shame(羞恥心)

(b) Joy または Distress との混合型
— Gratification(満足), Remorse(後悔)

(c) 他者の行動に関するもの
— Admiration(賞賛), Reproach(非難)

(d) Joy または Distress との混合型
— Gratitude(謝意), Anger(怒り)

3. 対象物に対する好き嫌いにのみ焦点を当てたもの
— Love(好き), Hate(嫌い)

ここでの混合型とは複数の感情が複合して生起する場合に生起する感情のことである。

3 従来研究

1 章で述べた通り、従来研究 [1, 2] では、全ての感情を実装するには至っていない。本章ではまず [1, 2] で実装されている感情の実装方法について述べ、次に [1, 2] で実装されていない感情については、実装する上での問題であった点を述べる。

3.1 実装に用いられた形式化

Adam らの形式化では、感情表現の生起条件を BDI logic での論理式で表現する。
①1(a)i：Joy, Distress
②1(a)iiA：Hope, Fear
③1(a)iiB：Satisfaction, FearConfirmed, 1(a)iiC：Relief, Disappointment
④1(b)：HappyFor, Resentment, SorryFor, Gloating
⑤2(a)：Pride, Shame, Gratification, Remorse
⑥2(b)：Admiration, Reproach, Gratitude, Anger
⑦3：Love, Hate
以上のように 22 種類の感情を 2 章の分類での 7 種類に分けると、それぞれ生起条件が同じ形で出てくる。

本節では、生起条件が同じのものについては、それぞれの中から一つの感情について、従来研究で実装されていた感情 (①〜⑤) の実装方法について述べる。

3.1.1 *Joy*

Joy(喜び) の生起条件は

$$Joy_i\varphi \equiv Bel_i\varphi \wedge Des_i\varphi$$

と記述される。

ここで、$Bel_i\varphi$ は「i は事象 φ の成立を信じている」、$Des_i\varphi$ は「i にとって事象 φ は望ましい」という意味で、上記の *Joy* の生起条件は「エージェント i が、事象 φ の成立を

信じ、かつそれが i にとって望ましいことであるならば、i は φ の発生に対して喜びという感情を生起する」という意味になる。基本的には、この式の右辺が成立するかどうかをJason で表現し、成立すれば *Joy* という感情が生起されたとする。

$Bel_i\varphi$ の表現は Jason での信念ベースをそのまま使用できるが、ここでの $Des_i\varphi$ は BDI モデルでの Desire(願望) とは別物で、Desirability(i にとって φ は望ましい) を表すため、Jason の願望つまり目標にあたる Goal を用いて表現することが出来ない。

Adam らの形式化では、

$$Des_i\varphi \Leftrightarrow Bel_iDes_i\varphi$$

が成り立つので、$Des_i\varphi$ であることと i がこれを信念に持つことは同等となり、$Des_i\varphi$ を表現する場合、i の信念ベースにこれを追加すればよい。従って、実装は、信念ベースに φ と $Des_i\varphi$ があるかどうかを検査し、条件が揃えば $Joy_i\varphi$ を結論する、という実装方針が取られている。

3.1.2 *Hope*

Hope(望ましい) の生起条件は

$$Hope_i\varphi \equiv Expect_i\varphi \wedge Des_i\varphi$$

と記述される。これは「i が事象 φ が成立しそうだと思っており、かつそれが i にとって望ましいことであるならば、i は φ の発生に対して望ましいという感情を生起する」という意味になる。

$Expect_i\varphi$ は $Prob_i\varphi \wedge \neg Bel_i\varphi$ の略記として定義され、$Prob_i\varphi$ は「φ である可能性が高い」という意味である。従って、$Expect_i\varphi$ は「φ を完全には信じていないが、ありそうであると思っている」という意味になる。

Prob は確度の低い信念とみなせるので、Jason では annotation 付き信念で表現されており、あとは *Joy* と同様に実装されている。

3.1.3 *Satisfaction*

Satisfaction(満足) の生起条件は

$$Satisfaction_i\varphi \equiv Bel_i\mathbf{P}Expect_i\varphi \wedge Des_i\varphi \wedge Bel_i\varphi$$

と記述される。これは「i が事象 φ の成立を過去のある時に期待し、かつ現在その成立を信じ、それが i にとって望ましいならば、i は φ が達成したということに対し満足している」という意味になる ($\mathbf{P}$ は「過去のある時」を表す)。しかし、この式を実装するには $Bel_i\mathbf{P}Expect_i\varphi$ の実現が難しいため、

$$Expect_i\varphi \wedge Des_i\varphi \supset \mathbf{A}(Satisfaction_i\varphi\mathbf{N}Bel_i\varphi)$$

とパラフレーズしている。これは「i が φ の成立を期待し、それが i にとって望ましいなら、次に i がその成立を信じたときに i は満足する」という意味となる ($\mathbf{A}(\psi\mathbf{N}\varphi)$ は「現時刻以後

最初に φ が成り立ったとき ψ も成り立つ」を表す)。Adam らの形式化では、Des は同じエージェント i に対しては時間変化によって変わらないものとされている ($Des_i\varphi \Leftrightarrow \mathbf{G}Des_i\varphi$ が成り立つ。ここで $\mathbf{G}$ は「未来永劫に」を表す) ため、$Des_i\varphi$ を検査するタイミングはいつでもよいとされている。

3.1.4 *HappyFor*

HappyFor(共に喜ばしく思う) の生起条件は

$$HappyFor_{i,j}\varphi \equiv Bel_i\varphi \wedge Bel_iDes_j\varphi \wedge Des_iBel_j\varphi$$

と記述される。

これは、「『エージェント i にとって “エージェント j が事象 φ の成立を信じる” ことは望ましい ($Bel_iDes_j\varphi$)』かつ、『エージェント i は “エージェント j にとって事象 φ が望ましい” と信じている ($Des_iBel_j\varphi$)』かつ、『エージェント i は事象 φ の成立を信じている ($Bel_ij\varphi$)』」ならば、i は φ が起きたことを j のために喜ぶという意味になる。

他者の感情が自分の感情に影響を及ぼすと言うことは、自エージェントの感情について推論するには他エージェントの感情を知る必要があるということである。しかし、Jason では各エージェントの心的状態はそのエージェント内でのみアクセスでき、他エージェントから知ることは出来ないため、他エージェントの感情に依存するような感情を実装するには、他エージェントの信念が変わる度にそれをこちらに知らせることが必要になる。他エージェントの信念を取得すること自体は、エージェント間の通信で実現できるが、“他者の感情を知るのに必要な情報” のみを “信念が変わる度に” 知らせるのをどのように実装するかが問題となっていた。

そこで、複数のエージェント間で通信によって他エージェントの感情を自分の信念として取得し、これを用いて自分の感情を生起出来るように実装されている。複数のエージェント間の通信方法としては、Jason の内部アクションを使用し、特定の信念について、信念追加イベントが発生した際に、同時に他エージェントに信念を送るようにして、その信念が他者に伝わるように実装されている。

3.1.5 *Pride*

Pride(自尊心) の生起条件は

$$Pride_i(i:\alpha,\varphi) \equiv Bel_iDone_{i:\alpha}(Idl_iHappens_{i:\alpha}\varphi \wedge Prob_iAfter_{i:\alpha}\neg\varphi) \wedge Bel_i\varphi$$

と記述される。

ここで、$Done_{i:\alpha}$ は「i がアクション α を実行する前は φ が成立していた」、$Idl_i\varphi$ は「φ が成立することは i にとって理想的である」、$Happens_{i:\alpha}\varphi$ は「i がアクション α を実行すると φ が成立する可能性がある」、$After_{i:\alpha}\varphi$ は「i がアクション α を実行すると φ が成立する」という意味である。

従って、*Pride* の生起条件の意味は、「『エージェント i がアクション α を実行して、事象 φ が成立することは i にとって理想的であり、かつ、i が α を実行すると φ が成立しな

い可能性が高いと思われていた ($Bel_i Done_{i:\alpha}(Idl_i Happens_{i:\alpha}\varphi \wedge Prob_i After_{i:\alpha}\neg\varphi)$)』かつ、『エージェント i は事象 φ の成立を信じている ($Bel_i\varphi$)』となり、大まかには「達成が困難だと思われていたことを達成することでエージェント i が事象 φ の達成を誇りに思う」という意味になる。

Adam らの形式化では、Idl は $Idl_i Happens_{i:\alpha}\varphi$ の形でしか現れないので、これを i の信念ベースに $Idl\_Happens(\alpha,\varphi)$ の形で持っておけば、別途 Idl オペレータを考慮する必要がなく、同等の情報が持て、$Idl_i Happens_{i:\alpha}\varphi$ と $Prob_i After_{i:\alpha}\neg\varphi$ の両方が得られれば $Idl_i Happens_{i:\alpha}\varphi \wedge Prob_i After_{i:\alpha}\neg\varphi$ が結論できる。

そこで、この感情を実現するためには、「$Idl_i Happens_{i:\alpha}\varphi \wedge Prob_i After_{i:\alpha}\neg\varphi$ が得られ、かつその後アクション α を実行して信念 φ が得られれば $Pride_i(i:\alpha,\varphi)$ を結論する」とすればよい。よって $Idl_i Happens_{i:\alpha}\varphi$ と $Prob_i After_{i:\alpha}\neg\varphi$ を信念として持った上で、信念 φ が得られれば $Pride$ が生起するので、感情の生起に関する行動 α のプランの後に信念 φ が得られたかどうかを検査し、生起条件が揃っていれば感情を生起し、それに伴った行動を取るように実装されている。

3.2 実装上の問題点

3.2.1 他者の行動に対する賞賛度に焦点を当てた感情

従来研究では、他者の行動に対する賞賛度に焦点を当てた感情 (3.1 の ⑥) が未実装であった。

例えば $Admiration$(賞賛) の生起条件は

$$Admiration_{i,j}(j:\alpha,\varphi) \equiv Bel_i Done_{j:\alpha}(Idl_i Happens_{j:\alpha}\varphi \wedge Prob_i After_{j:\alpha}\neg\varphi) \wedge Bel_i\varphi$$

と記述される。

この生起条件の意味は「『エージェント j がアクション α を実行して、事象 φ が成立することは i にとって理想的であり、かつ、j が α を実行すると φ が成立しない可能性が高いと i は思っていた ($Bel_i Done_{j:\alpha}(Idl_i Happens_{j:\alpha}\varphi \wedge Prob_i After_{j:\alpha}\neg\varphi)$)』かつ、『エージェント i は事象 φ の成立を信じている ($Bel_i\varphi$)』となる。

従来研究では、$Idl_i Happens_{i:\alpha}\varphi$ と $Prob_i After_{i:\alpha}\neg\varphi$ の扱いについては、エージェント i が $Pride$ が生起する条件である信念 φ のリストを作り、そのリストと現在保持している信念を比較し、感情を生起していた。だが、任意のタイミングでの信念追加には対応していなかった。また、$Idl_i Happens_{i:\alpha}\varphi$ と $Prob_i After_{i:\alpha}\neg\varphi$ は φ のみ引数として任意に指定できるという形式で表現していたため、$Admiration$ の定義にあるように j のアクションにより φ が成り立つということを表すことは出来なかった。

3.2.2 混合型

従来研究では、混合型の感情が未実装であった。

例えば $Gratification$(満足) の生起条件は

$$Gratification_i(i:\alpha,\varphi) \equiv Pride_i(i:\alpha,\varphi) \wedge Joy_i\varphi$$

と記述される。

従来研究ではそれぞれ *Joy* と *Pride* は実装されていた。だが別の研究で扱ったものであり、Bel, Des の定義形式が異なっていた。*Joy* の生起条件では事象 α のみを引数にしてどのエージェントの信念であるかには触れていないのに対して、*Pride* の生起条件ではエージェント i の信念であるということを条件として指定していた。つまり定義形式を統一しないと生起条件となる信念、願望を複数の形式で定義しなければならないという問題があった。

4 実装方法

3 章で述べたように、従来研究 [1, 2] ではいくつかの感情が未実装であった。本章では、未実装の感情の実現のため、その解決方法と実装方針について述べる。生起条件が同じ形の感情については、それぞれの中から 1 つの感情について、本研究で実装した感情 (3.1 の⑤,⑥) の実装方法について述べる。

4.1 実装に用いる形式化

実装に用いる形式化は従来研究と同様、OCC theory における感情の生起条件を BDI logic の論理式で Adam らが形式化したものであるが、それを Jason の規則として記述する上で、従来の問題点を解決するよう変更し、プランの前提条件として使用出来るようにし、感情生起の実現と感情を行動決定に用いる事が出来るように実装した。実装に用いる形式化については、3 章で述べた 3.2.1 と 3.2.2 を用いた。

4.1.1 *Admiration*

Admiration(賞賛) の生起条件は

$$Admiration_{i,j}(j:\alpha,\varphi) \equiv Bel_i Done_{j:\alpha}(Idl_i Happens_{j:\alpha}\varphi \wedge Prob_i After_{j:\alpha}\neg\varphi) \wedge Bel_i\varphi$$

と記述される。

この生起条件の意味は、3.2.1 で記述したように「『エージェント j がアクション α を実行して、事象 φ が成立することは i にとって理想的であり、かつ、j が α を実行すると φ が成立しない可能性が高いと i は思っていた $(Bel_i Done_{j:\alpha}(Idl_i Happens_{j:\alpha}\varphi \wedge Prob_i After_{j:\alpha}\neg\varphi))$』かつ、『エージェント i は事象 φ の成立を信じている $(Bel_i\varphi)$』ならば i は j が φ を達成したことを賞賛するという意味である。

この感情を実現するためには従来研究での $Idl_i Happens_{i:\alpha}\varphi$ と $Prob_i After_{i:\alpha}\neg\varphi$ の定義形式を変更し、*Admiration* の生起条件を「エージェント j が φ を達成した時」として定義することで実現できる。

4.1.2 *Gratification*

Gratification(満足) の生起条件は

$$Gratification_i(i:\alpha,\varphi) \equiv Pride_i(i:\alpha,\varphi) \wedge Joy_i\varphi$$

と記述される。

この感情の生起条件は *Pride* と *Joy* が生起していることである。*Pride* と *Joy* の生起条件は

$$Pride_i(i:\alpha,\varphi) \equiv Bel_i Done_{i:\alpha}(Idl_i Happens_{i:\alpha}\varphi \wedge Prob_i After_{i:\alpha}\neg\varphi) \wedge Bel_i\varphi$$

$$Joy_i\varphi \equiv Bel_i\varphi \wedge Des_i\varphi$$

である。つまりエージェント i にとって誇りが持てるような事象 φ が達成でき、かつそれが喜ばしいことであるとき、i は満足するという意味である。

この感情を実現するためには Bel, Des の定義形式を変更し、エージェント i の信念 φ であるというようにどのエージェントの信念、願望であるかを明示するように統一することで実現できる。

4.2 実装方針

感情生起に関するプログラムは Jason 上でエージェントを記述する asl ファイルとして実現されており、3 つのファイルからなる。em.asl は感情の生起条件のみを定義したものであり、pride-shame.asl には *Pride*, *Shame*, *Admiration*, *Reproach* の生起に必要なリスト制作や検査に必要なプランが記述されている。con.asl には通信や追加された信念、ゴールに関するプランが記述されている。これらをそれぞれのエージェントに include することにより感情生起が扱えるようになっている。

ここではこれら 3 つのプログラムとそれぞれのエージェントの実装について述べる。

4.2.1 他者の行動に対する賞賛度に焦点を当てた感情

3.2.1 で述べたように、他者の行動を評価するような感情の実装においては、自身の行動を評価する感情の実装と同じようにエージェントが感情生起の条件となる信念 φ のリストを持つように実装する必要がある。だが自身の行動評価の場合とは異なり他者の行動を判断するので、どのエージェントの信念であるかを定義しておかなければならない。よって、エージェント j が信念 φ を得られた時に *Admiration* を生起するとすると、*Admiration* の生起条件は “j が φ を得た時” と扱い、これら 2 つの情報をセットにしてリスト化するように実装した。

実際の実装形式は図 1 のようになる。

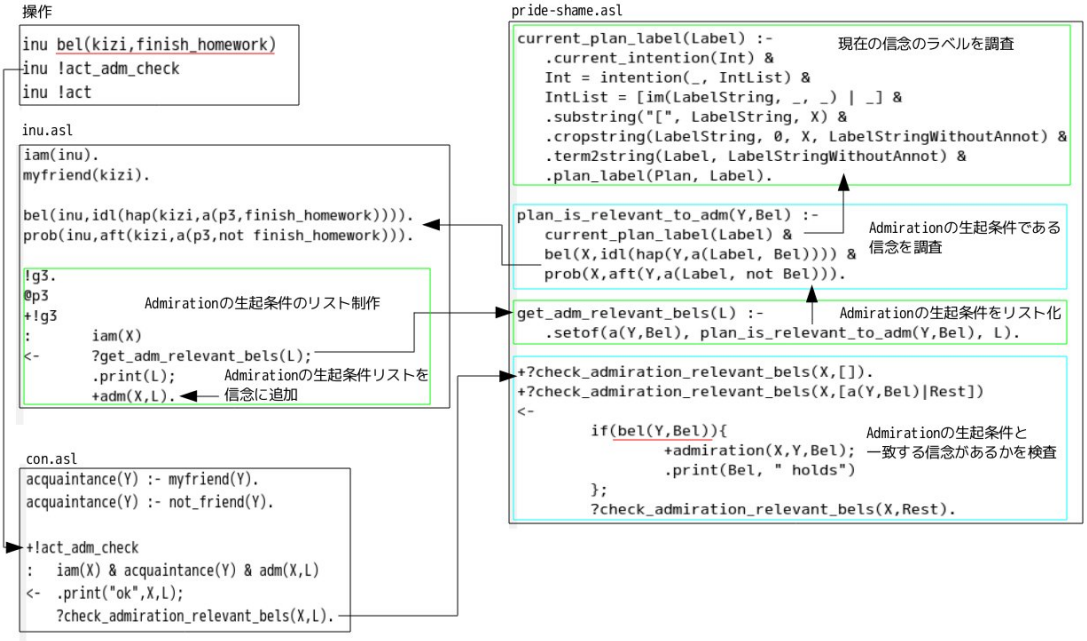

図 1: *Admiration*

4.2.2 混合型

混合型の感情の実装においては Bel,Des の定義形式を統一した。従来研究では信念を事実として記述していたが、他エージェントとの通信を必要とする感情もあるのでどのエージェントの信念かを明確にするため、エージェント i が X という信念を持つときは $Bel(i, X)$ と記述するようにした。Des,$Prob$ などを含む全ての感情も引数を 2 つ以上持つ。このように形式を統一することで感情生起のためにエージェントが必要とする信念、願望を複数の定義形式で定義する必要がなくなり、感情生起のための信念、願望の追加が容易になった。

また、混合型には他者の行動が感情生起の条件となっている感情もあるので、従来の Joy の様に自エージェントの行動に関する願望だけでは生起しない。よってこの場合、Joy の生起条件はどのエージェントの行動の結果であっても φ が成り立っていれば良いということで $Des(anyone, X)$ という形で定義した。この時の anyone は定数となっている。

よって事象 φ によって行動に対する賞賛度に関する感情が生起した時、φ がエージェントにとって望ましいことであるなら Joy も生起する。混合型の生起条件である 2 つの感情がどちらも成り立った場合に混合型の感情を生起し、それに伴った行動を取るように実装した。

実際の実装形式は図 2 のようになる。

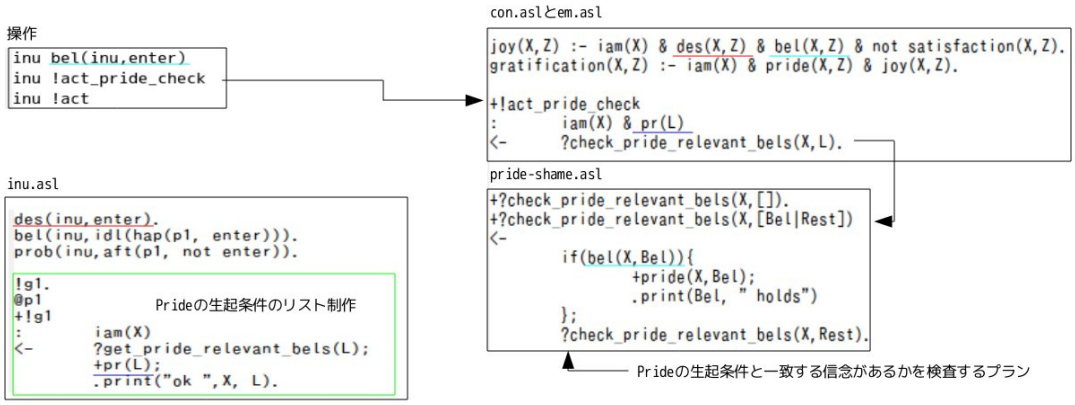

図 2: *Gratification*

4.2.3 ライブラリ化

ライブラリ化において変更した点を述べる。

Expect

Expect を定義に含む感情は

$$Hope, Fear, Satisfaction, FearConfirmed, Relief, Disappointment$$

である。これらは従来研究の手法ではあらかじめシステム内に記述している事象でなければ扱えないという欠点があった。

この欠点を解決する手段として、単純に任意の信念を追加するプラン、つまり従来研究では指定されている事象を任意の事象とした場合の問題点としては以下の 2 点が挙げられる。

1. 確度を伴った信念を外部から入力した後にエージェント自身がその情報を元に信念を追加しようとすると、情報源に Jason が annotation を付加する際にその付加する位置がおかしくなり、該当するはずのプランが選ばれなくなる。

2. 任意の事象とすることで意図しないタイミングでプランが使われる可能性がある。

これらを回避するため、外部からの入力を plus(X)[degOfCert(Y)] という形に統一し、Y の値によって追加する信念を分類した。これにより *Prob*, *Expect* が適切に追加され、*Satisfaction* などで用いられる *exist_expect* という述語が正常に機能する。

この plus 述語を用いることによりシステム内に固有名詞をあらかじめ記述しておかなければならないという欠点は解消され、より汎用的になった。

HappyFor

HappyFor など他エージェントとの通信が必要な感情については、相手が友好的な関係のものであるか敵対する関係のものであるかと、良い知らせなのか悪い知らせなのかという点において、通信により追加される信念の処理方法を分類し、記述した。

これらはエージェント同士の関係性と、エージェントの願望という形でそのエージェントにとってよい情報か否かをあらかじめ記述しておくことにより判断が可能であり、その事象自体がどのようなものであるかは考慮しなくてよい。よってこれら 2 つの情報があれば任意の事象について扱える。

5 実験と結果

本章では、実装した感情が期待通りに生起され、エージェントのプランの選択に影響を与えることを実験で示す。また、ライブラリ化したことにより既存のエージェントプログラムに感情の生起を判定するテストゴールを追加することで、容易に感情表現を導入できることを示す。

5.1 実験

サンプルとして、複数のシナリオを作り、そのシナリオ通りにエージェントを動かすことで、シナリオに沿った感情を生起させ、生起された感情によって行動を決定するようなプランを Jason 上に記述し、期待通りの感情生起と行動が起こることを確認した。

またそれに伴い、エージェントに対し外部から任意のタイミングで信念の追加を行うイベントや、ゴールの追加を行うイベントを与える必要があるので、外部から TCP の 49999 番ポートに接続すると、指定したエージェントにこれらのイベントを与えられるような環境を用意した。

5.2 結果

5.2.1 *Relief*

Relief (安堵) を生起させるシナリオとして、以下のように設定する。

エージェント inu と kizi がおり、この 2 人は友達である。4 月のクラス替えの際、inu は友達と同じクラスになりたいという願望を持っていた (des(inu,friend_same_class))。inu は可能性の低いことだと思っていたが 2 人は同じクラスになり、inu は安堵する。

このシナリオは図 3 のようなプラン (一部抜粋) によって実装される。plus 関数により受け取ったものを Y の値によって場合分けし、それぞれに応じた信念を追加するようになっている。確度の低いものであったとき、expect_not(X,Z) が成り立つので、*Relief* の生起条件の一つである exist_expect_not(X,Z) が信念に追加される。

操作としては図 4 のように実行する。*Relief* は起こると予想していなかった望ましい出来事が実際に起こったときに生起する感情である。よって予想が必要となるので plus という関数を用いてエージェントに信念を追加する。この信念は予想するためのものである

ので、エージェントが予想を信念として追加した後に削除する。その後実際に出来事が起こったと知ると *Relief* を生起する。

結果としては図 5 のようになり、期待通りの感情生起と行動決定を取ることが出来た。

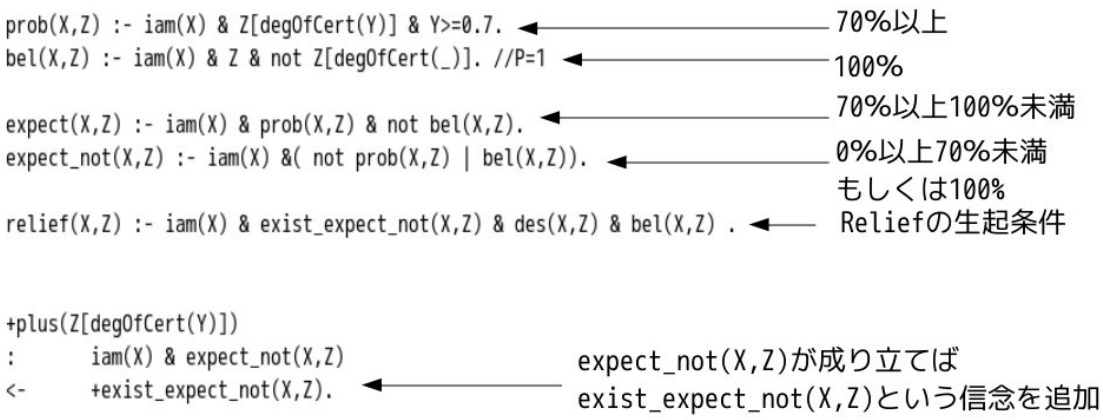

図 3: *Relief* を生起させるプラン

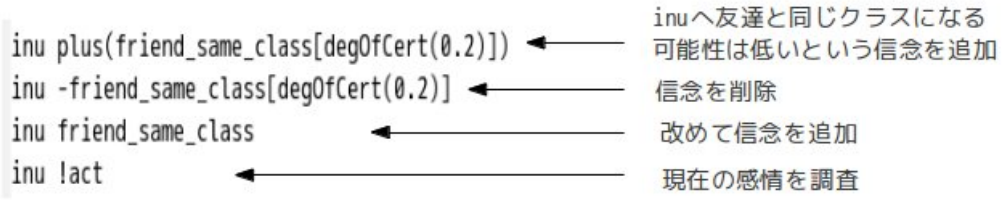

図 4: *Relief* を生起させる操作

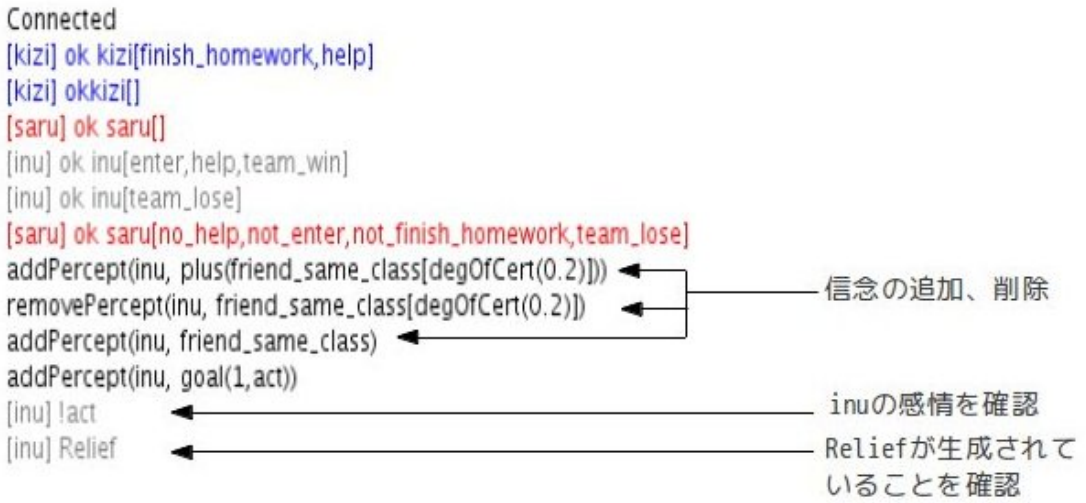

図 5: *Relief* を生起させた結果

5.2.2 *HappyFor*

HappyFor(共に喜ばしく思う) を生起させるシナリオとして、以下のように設定する。

エージェント inu と kizi がおり、この 2 人は友達である。1 月の受験の際、inu は kizi に合格したいと思っていることを伝える。その後 inu が kizi に合格したいということを伝えると kizi は *HappyFor* を生起する。

このシナリオは図のようなプラン (一部抜粋) によって実装される。!act_tof_goodex は友好関係にあるエージェントに自分の願望を伝えるためのものである。受け手の友好関係にあるエージェントはその信念を追加した上で、その信念が成り立つことは自分にとっても望ましいことであるという信念を追加する。!act_toa_belx は他のエージェントに自分の信念を伝えるためのものである。これらにより願望、信念を受け取ったエージェントは *HappyFor* の生起条件を満たし、生起する。

操作としては図 7 のように実行する。inu は友達である kizi に合格したいという願望を伝える。その後 inu は合格するという信念を得る。inu が kizi に合格したという信念を伝えると kizi は共に喜ばしく思う。

結果としては図 8 のようになり、期待通りの感情生起と行動決定を取ることが出来た。

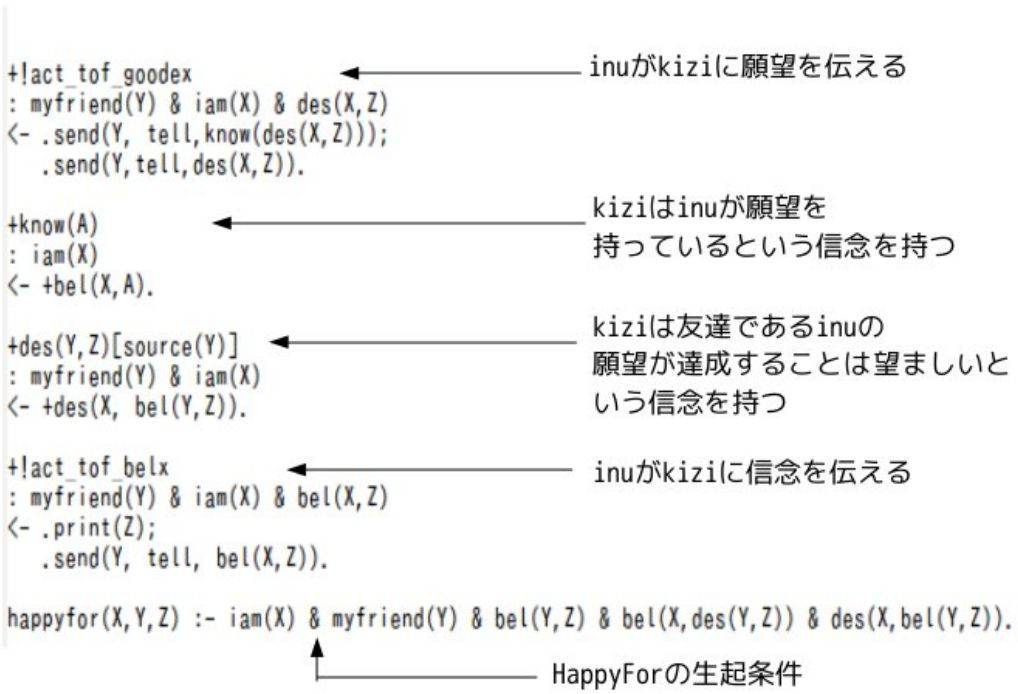

図 6: *HappyFor* を生起させるプラン

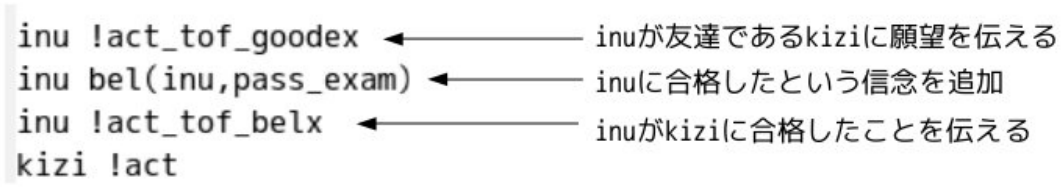

図 7: *HappyFor* を生起させる操作

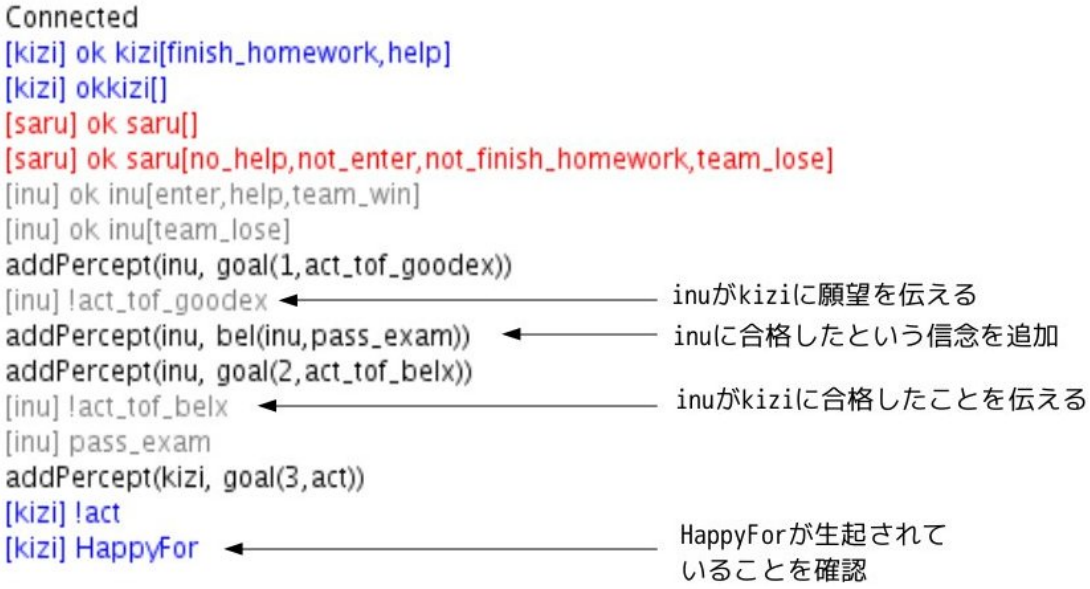

図 8: *HappyFor* を生起させた結果

5.2.3 *Admiration*

Admiration(賞賛) を生起させるシナリオとして、以下のように設定する。エージェント inu と kizi がおり、この 2 人は友達である。8 月の夏休みに inu は kizi が宿題を終わらせていることを知ると inu は kizi に対して *Admiration* を生起する。

このシナリオは図 9,10,11 のようなプラン (一部抜粋) によって実装される。エージェント i はエージェント j が α をした時に賞賛するとする。i の信念に予め bel(i,idl(hap(j,a(p3,α)))) と prob(i,aft(j,a(p3,not α))) と記述しておく。*Pride* の定義形式と異なっているのは行動をするエージェントを定義するために定義を細分化したためである。エージェントはこの信念を基に [i,[(a,j_1,α_1),a(j_2,α_2))]] という風にリストを生成する。これはエージェント i はエージェント j_1 が α_1 を達成した時や j_2 が α_2 を達成した時に *Admiraion* を生起するという意味になる。このリストと i が保持している信念を比較し、該当する信念を見つけたときに *Admiration* を生起する。

操作としては図 12 のように実行する。inu は kizi が宿題を終わらせたという信念を得る。inu は自分の *Admiration* の生起条件のリストと得られた信念を比較し、一致するかを調べる。そうすると一致する信念があるので、inu は kizi を賞賛する。

結果としては図 13 のようになり、期待通りの感情生起と行動決定を取ることが出来た。

```
iam(inu).                                          ← エージェントの関係性
myfriend(kizi).

bel(inu,idl(hap(kizi,a(p3,finish_homework)))).     ← inuがAdmirationを生起する条件
prob(inu,aft(kizi,a(p3,not finish_homework))).

!g3.
@p3
+!g3                                               ← Admirationの生起条件のリスト制作
:       iam(X)
<-      ?get_adm_relevant_bels(L);
        .print(L);
        +adm(X,L).
```

図 9: *Admiration* を生起させるプラン (inu.asl)

```
current_plan_label(Label) :-
   .current_intention(Int) &
   Int = intention(_, IntList) &
   IntList = [im(LabelString, _, _) | _] &
   .substring("[", LabelString, X) &
   .cropstring(LabelString, 0, X, LabelStringWithoutAnnot) &
   .term2string(Label, LabelStringWithoutAnnot) &
   .plan_label(Plan, Label).

plan_is_relevant_to_adm(Y,Bel) :-
   current_plan_label(Label) &
   bel(X,idl(hap(Y,a(Label, Bel)))) &
   prob(X,aft(Y,a(Label, not Bel))).

get_adm_relevant_bels(L) :-
   .setof(a(Y,Bel), plan_is_relevant_to_adm(Y,Bel), L).

+?check_admiration_relevant_bels(X,[]).
+?check_admiration_relevant_bels(X,[a(Y,Bel)|Rest])
<-
        if(bel(Y,Bel)){
                +admiration(X,Y,Bel);
                .print(Bel, " holds")
        };
        ?check_admiration_relevant_bels(X,Rest).
```

現在の意図のラベルを調査

Admirationが生起する条件を調査

Admirationの生起条件をリスト化

Admirationの生起条件のリストと一致する信念があるかを検査

図 10: *Admiration* を生起させるプラン (pride-shame.asl)

```
acquaintance(Y) :- myfriend(Y).
acquaintance(Y) :- not_friend(Y).

+!act_adm_check
:   iam(X) & acquaintance(Y) & adm(X,L)
<-  .print("ok",X,L);
    ?check_admiration_relevant_bels(X,L).
```

関係性を重視しないプランのための規則

Admirationを生起するかを検査

図 11: *Admiration* を生起させるプラン (con.asl)

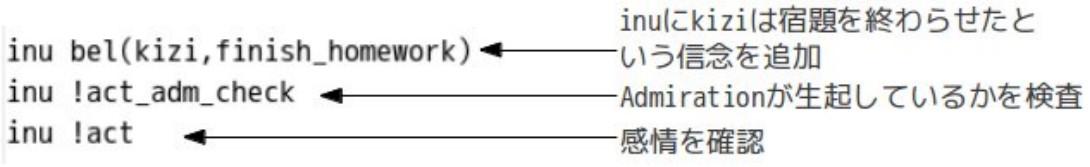

図 12: *Admiration* を生起させる操作

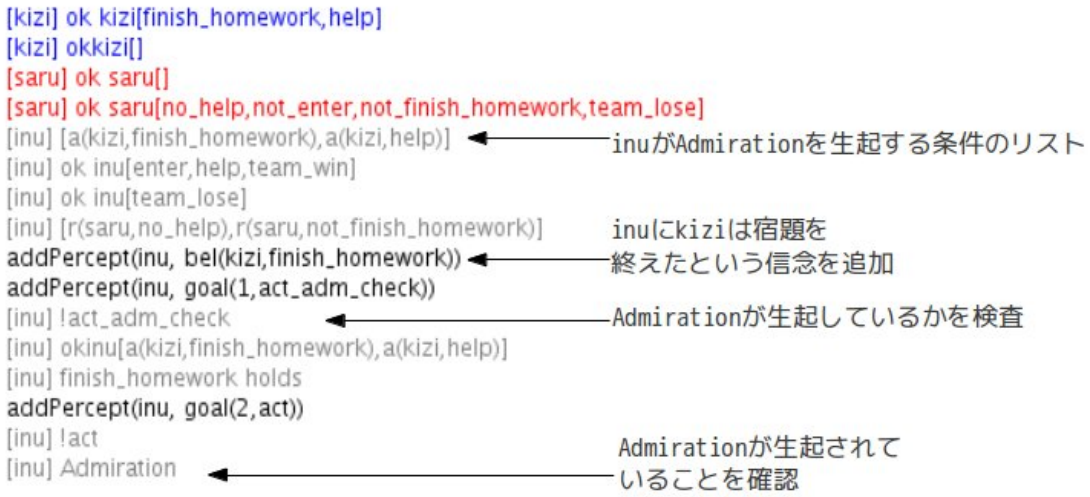

図 13: *Admiration* を生起させた結果

5.2.4 *Gratification*

Gratification(満足) を生起させるシナリオとして、以下のように設定する。エージェント inu がおり、3 月の卒業を控えている。inu は進学したいと思っているし、進学できたら誇りが持てると思っている。そして進学できた inu は *Gratification* を生起する。

このシナリオは図 14,15 のようなプラン (一部抜粋) によって実装される。*Gratification* は複合型であり、*Pride* と *Joy* の 2 つが生起条件となる。

Pride は従来研究では任意のタイミングのイベントに対応していなかった。そこでエージェントそれぞれに *Pride* を生起する出来事のリストを最初に保持させる。その上で信念を任意のタイミングで追加し、その後 *Pride* が生起するか否か、つまり現在保持している信念とリストに一致するものはあるかを調べさせ、一致するものがあった場合 *Pride* を生起するようにした。

そして *Pride* が生起するための信念が自分にとって望ましいものであるかを調べ、望ましいものであったときは *Joy* を生起する。これら 2 つの感情が成り立った場合に *Gratification* を生起する。

操作としては図 16 のように実行する。inu に進学できるという信念を追加し、*Pride* が生起するかを調べさせる。結果 *Pride* が生起し、かつ *Joy* も成り立つので inu は満足する。

結果としては図 17 のようになり、期待通りの感情生起と行動決定を取ることが出来た。

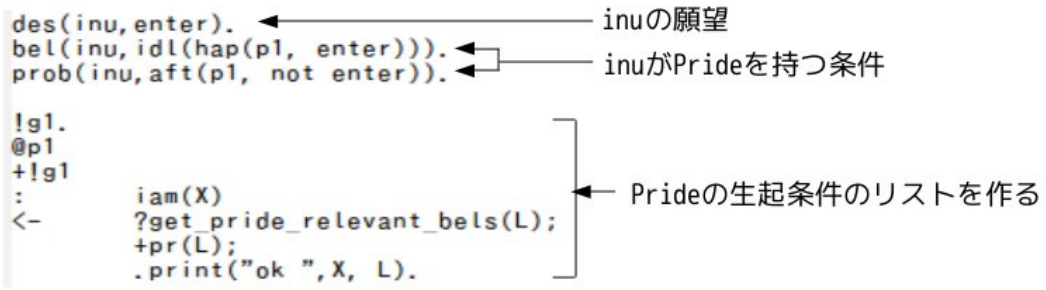

図 14: *Gratification* を生起させるプラン (inu.asl)

```
joy(X,Z) :- iam(X) & des(X,Z) & bel(X,Z) & not satisfaction(X,Z).
gratification(X,Z) :- iam(X) & pride(X,Z) & joy(X,Z).   ← Gratificationの生起条件

+!act_pride_check                    ← Prideが生起しているかを検査
:       iam(X) & pr(L)
<-      ?check_pride_relevant_bels(X,L).


+?check_pride_relevant_bels(X,[]).   ← Prideが生起するかを調べるプラン
+?check_pride_relevant_bels(X,[Bel|Rest])   該当する信念があれば
<-                                          Prideを信念に追加
        if(bel(X,Bel)){
                +pride(X,Bel);
                .print(Bel, " holds")
        };
        ?check_pride_relevant_bels(X,Rest).
```

図 15: *Gratification* を生起させるプラン

```
inu bel(inu,enter)      ← inuに信念を追加
inu !act_pride_check    ← Prideが生起するかを検査
inu !act
```

図 16: *Gratification* を生起させる操作

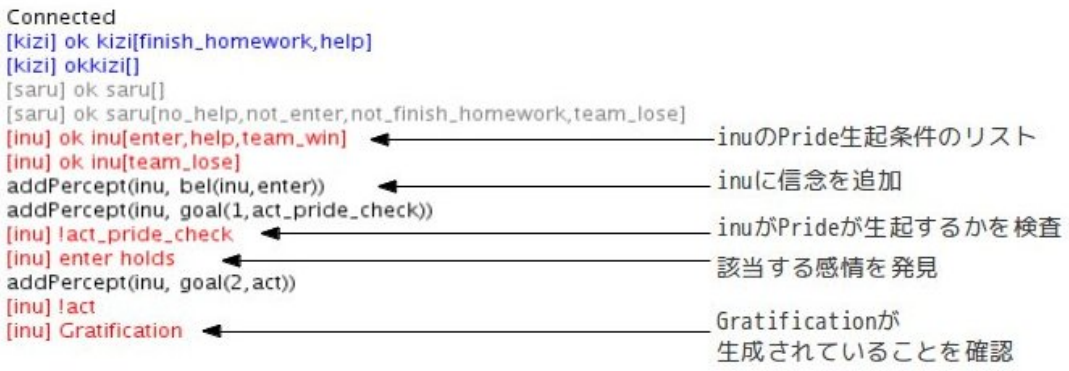

図 17: *Gratification* を生起させた結果

5.3 導入例

本研究で実装したプログラムが汎用的に使えることを示すため、Jason の例題のプログラムに感情表現を導入した。

導入したプログラムは Jason がサンプルとして用意している DomesticRobot というプログラムである。

エージェント robot,owner,supermarket があり、robot は owner にビールを運ぶ。owner はビールを飲み、なくなったら robot に冷蔵庫まで取りに行かせる。robot は冷蔵庫内にビールが残っていたら owner に運び、残っていなかったら supermarket に買いに行った後、owner に運ぶ。そして owner がビールを 10 杯飲んだ時に robot は止まる。

このプログラムを変更し、robot は冷蔵庫にビールが無かった場合、supermarket に買いにいくことを苦痛 (*Distress*) に思い、仕事をサボる。それを見た owner は怒り (*Anger*) を感じる。

プログラムの変更箇所は図 18,19 のようになっている。robot にとって supermarket に買いに行くことは望ましくないという信念と、owner には robot がサボることは理想的ではない、だが確率は低いという信念と、かつサボることは望ましくないという信念を追加した。また、感情生起に必要なプラン、つまり信念追加や他エージェントとの通信も追加した。

結果としては図 20 のようになり、期待通りの感情生起と行動決定を取ることが出来た。よって既存のエージェントに本研究のプログラムを用いて信念やプランを追加すれば、感情表現を持つエージェントが実装できる。

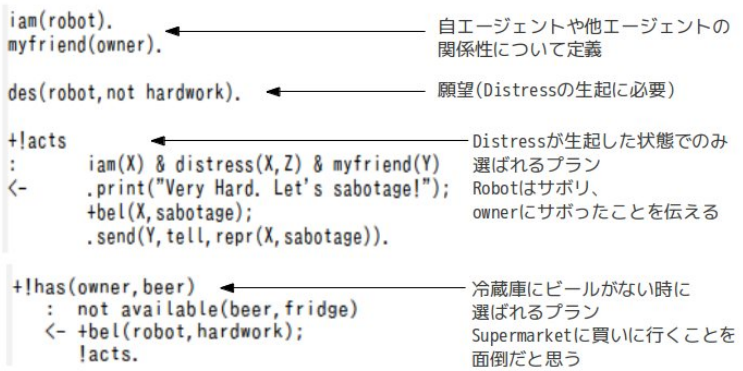

図 18: robot.asl の変更箇所

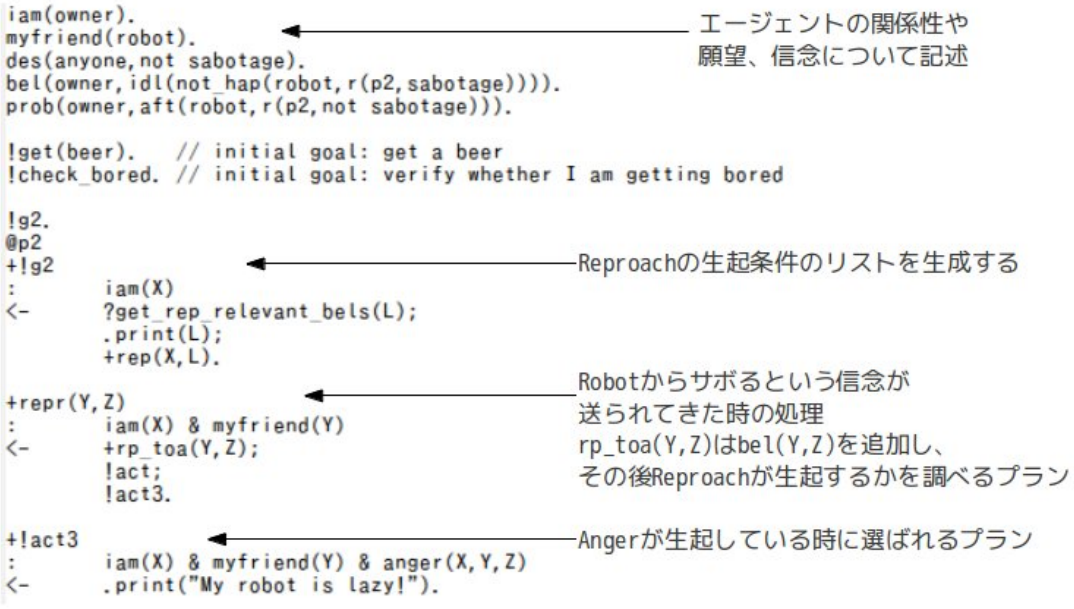

図 19: owner.asl の変更箇所

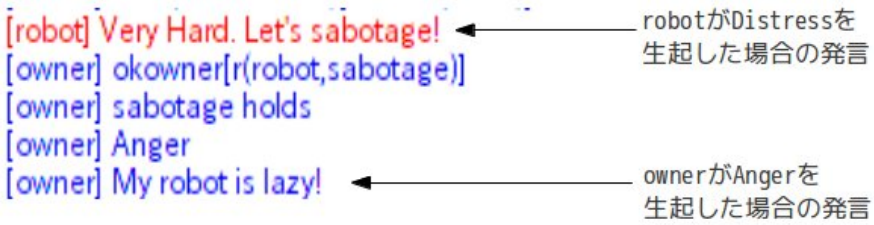

図 20: 実行結果

6　終わりに

以上の結果より、あるシナリオにおいて、エージェントは OCC theory に基づいて形式化された感情を推論して生起し、またそれに従った行動を取ることができ、従来の研究では実現されていなかった新たな感情を実現した。

今後の課題としては、本研究で実装した感情の妥当性の評価がある。本研究では、3.1 に述べた Adam らによる形式化をそのまま使用しているため、今回実現した感情の妥当性も Adam らの形式化に依存することになるが、その妥当性も含めて今後議論する必要がある。

また、複数の感情の競合をどのように処理するか、対立関係にある感情の同時発生の扱いや、複合的な感情をどのように定義するかが問題となっている。さらに現在、生成された感情は時間が経過しても消えないので改良し、より人間に近いエージェントの実現を目指すことも必要である。

謝辞

本研究及び本論文の執筆にあたり、指導教官の新出尚之准教授には丁寧なご指導、ご助言を賜りました。心からの感謝の気持ちとお礼を申し上げたく、謝辞にかえさせていただきます。

参考文献

[1] 清水詩子. 感情表現を用いた行動決定を行うエージェントの実現. 奈良女子大学理学部情報科学科 2011 年度卒業論文, 2012.

[2] 池之内彰子. OCC theory に基づくエージェントの感情表現の改良. 奈良女子大学理学部情報科学科 2012 年度卒業論文, 2013.

[3] Carole Adam, Andreas Herzig, and Dominique Longin. A logical formalization of OCC theory of emotions. *Synthese*, Vol. 168, No.2, pp.201-248, 2009.

[4] Patricia Augustin Jaques and Rosa Maria Vicari. A BDI approach to infer student's emotions in an intelligent learning enviroment. *Comput. Educ.*, Vol.49, pp.360-384, September 2007.

[5] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. Programming Multi-Agent Systems in AgentSpeak using Jason. John Wiley & Sons, 2007.

[6] A. Ortony, G.L. Clore, and A Collins. *The Congnitive Structure of Emotions.* Cambridge University Press, 1988.

[7] Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff. Formal method in DAI. *Multi-agent Systems: a Modern, Approach to Distributed Artificial Intelligence,* 1999.

[8] 藤田恵, 片山寛子, 新出尚之, 高田司郎. 実世界の多様性に適応したロボットについて. 情報処理学会論文誌 数理モデル化と応用, 2012.