織田信長: 優先度に基づく並行論理型言語

平田圭二 (NTT CS 基礎研究所)　山崎憲一 (NTT 未来ねっと研究所)
hirata@brl.ntt.co.jp　yamazaki@t.onlab.ntt.co.jp

1 はじめに

論理型言語や並行論理型言語 (Concurrent Logic Language, CLL) によるプログラミング研究の目標の 1 つは, 何を計算するかということだけを記述し, どう計算するかということを記述せずに済むような枠組を構築することである. この枠組は一般に宣言的プログラミングとも呼ばれている. CLL としては, これまで GHC, Parlog などのプログラミング言語が設計・開発されており, この目標に対しある程度の成功を収めている. しかし, より実用的な並列分散処理を考えた場合, どう計算するかという実行制御の情報までプログラマが積極的に与えないと記述できない, あるいは記述するのが困難な場合が多く存在する. 例えば, 割り込み, 例外処理, 見込み計算, 実時間処理, 探索問題, ある種の分散アルゴリズムなど決定的に順序制御を行うような場合, プロセスの優先度という概念を用いてプログラマが CPU という資源を並行アクティビティに割り当て, 実行制御するのが普通である[1].

しかし CLL はその成り立ち故に優先度の概念を持っていない. このため, このような並列分散処理を CLL で記述し実現する方法は以下の 3 つに大別される. (1) 優先度に基づいて実行制御するメタレベルのプログラムモジュールを作成する. つまり, メタインタプリタを作成し, 優先度は既存の CLL のデータ構造を利用して表現する. (2) 用途に特化された組込み述語やシステムライブラリを提供しその内部で優先度を処理する. 例えば, 割り込み処理のためのメッセージ未着検出の述語 [7], 見込み計算のためのプロセッサのアイドル状態検出の述語 [3] などがある. (3) 言語及び実行モデルに優先度を取り込む. このアプローチは言語を理論的に扱いやすいという意味で優れた方法であり, 我々もこのアプローチを採用する.

(3) のアプローチを採る研究として KL1[10], Huntback の言語 [6] などがある. これらの言語は, 論理的解釈が可能な従来の CLL の部分と優先度に関する記述の部分から構成されている. この優先度の付加は元のプログラムの論理的意味を変更しないとしている. 同様に (2) の研究においても, 優先度を操作する構文要素は論理的に transparent であるとしている. 何故なら, 優先度を付加したプログラムを実行した時に得られる解の集合 P は, 優先度の無い元のプログラムを実行した時に得られる解の集合 O に含まれる ($P \subseteq O$) からである. KL1 では優先度に形式的な定義は与えられておらず, 実行に際しては, 優先度の順序をできるだけ満たすよう実行制御するということだけが要請されている (Huntback の言語でも形式的定義や実行規則に関する言及は見当たらない).

このように, 従来の研究は優先度の持つ論理的な意味には殆んど着目していなかった. しかし我々は, 優先度を付加することで, プログラムの振る舞いが変化し解が変化するのであるから, 優先度も何らかの論理的な意味を有していると考える[2].

我々の研究目標は以下の 2 つである. ● 優先度に基づく CLL を設計すること. 上述したように, 優先度を用いなければ記述できないあるいは記述困難な並列分散処理が存在している. 優先度に基づく CLL は, これらを記述できる程度の高い制御性を持つ必要がある. ● 優先度を形式化すること. これまで優先度は, 「どう計算するのか」ということを表現する手段として扱われて来た. この優先度を CLL の上で形式化することにより, 「何を計算するか」ということを記述する宣言的プログラミングに取り込むことができる.

本論文では並行論理型言語 **織田信長** を提案する. 織田信長は, 優先度を表現するための特別なデータ型を持ち, 優先度を一級オブジェクトとして取り扱うことができる. これより, 並列分散処理における柔軟な実行制御や汎用的なプログラミングが可能となる. 本論文では, 織田信長の設計方針, 構文, 実行規則を述べ, 幾つかのプログラム例を挙げ, 他の並列並行論理型言語や関連研究との比較検討を行う.

2 設計方針

我々は 織田信長を設計するにあたり次の点に留意した.

[1]本論文では, 例えば CPU 時間の 20% をプロセス A に, 80% をプロセス B に割り当てるというような問題は考えない.

[2]CLL ではないが [2] にも同様の主張がある.

(a) 優先度を含む項を述語として表現する

(b) 優先度に関するプログラマの意図を直感的かつ必要最小限に表現する

(c) 優先度によって 2 種類の非決定性を制御する

以下, 順に説明を加える.

(a) 論理型言語の特徴の 1 つは, すべての概念を**述語**によって表現, 記述することであり, これにより宣言的プログラミングが可能となっている. 述語として表現するとは, 一度確定した概念間の関係が対象としている世界において不変であることを意味している. これより, プログラムを論理式として解釈した時の意味が手続きとして解釈した時の意味に一致し (線形導出の健全性と完全性)[8], 並行論理型言語でも同様の結果が得られている [9]. このような性質が成り立つことで, 何を計算するのかという情報や性質をプログラムから静的に検出することができ, さらに, どう計算するのかという手順の効率化に役立てることが可能となる. 優先度を付加した項も述語として表現することで, 優先度を宣言的に記述でき, 宣言的プログラミングの利点を享受できるようになる.

(b) 従来の並列分散処理では優先度が自然数で与えられていた. これには以下のような得失がある.

利点 1 プロセスの個数が決まっている場合には, 自然数は順序付けの尺度として直感に合っている. しかし, 動的に生成される場合には, 必ずしもそうではない[3].

利点 2 自然数であることを利用したプログラミング技法が使える. 例えば, 優先度に対して加算や乗算等の算術演算を適用して, その結果をまた優先度として使うなどである.

欠点 1 プログラマが意図しなかった (余計な) 順序が付いてしまう場合がある. あるモジュール中のプロセスに優先度が割り振られていて, システム全体がモジュール群から構成されているとしよう. すると必然的に, 異なるモジュールに属するプロセスの優先度間にも順序が付いてしまう. この順序がプログラマが意図しなかった結果を招くこともある (図 7 参照).

欠点 2 優先度の値を決める際に曖昧性が存在する. 例えば, プロセス p を優先度 10 のプロセス q よりも低い優先度で実行したいとする. プログラマはこれを適当に 7 などと設定するが, これはプログラマへの負担となる.

このように優先度を自然数で表現することには大きな問題がある. 優先度を与えるデータ型は, プログラマの意図を直感的かつ必要最小限に表現するものが望ましい.

(c) CLL の実行モデルには一般に 2 種類の非決定性がある. 例えば下のような簡単な FGHC プログラムを考えよう.

```
:- X = a, Y = b, p(X,Y), q.

p(X,Y) :- X = a | ...   (1)
p(X,Y) :- Y = b | ...   (2)
q.
```

このプログラムでは, 変数 X, Y を介してプロセス p と通信が行われる. 変数 X に a, Y に b が代入され, p の定義中で変数 X, Y から値が読み出される. この実行過程は, p, q の述語呼出しと p の定義節における選択の 2 種類の操作から構成される. この時, 述語 p, q のどちらを先に呼出すか, 定義節 (1) (2) の内, どちらを選択するかという非決定性が生じている. 織田信長では, この述語呼出しと節選択における非決定性を優先度で制御する.

3 構文と実行規則

織田信長の構文と実行規則は FGHC を基にしているが, 主に次の 3 点で異なる.

- 全ての述語呼出しと能動単一化に優先度を付与する.
- 優先度間の大小関係だけを指定する. この指定によって優先度変数が具体化される.
- 各優先度変数に具体化を行う出現は高々 1 回である.

[3]例えば, 昔の BASIC の行番号のように優先度を 10 おきに使うといった情けないテクニックが必要となる.

まず, 述語呼出しだけでなく全ての単一化に優先度を付与するため, 全ての単一化を明示的に記述する必要がある. そこで 織田信長では, 述語引数の位置や関数引数の位置には相異なる変数のみ出現するという制限を加えた. 全ての単一化はガード部やボディ部に陽に記述しなければならない.

定義節は基本的に

$$\underbrace{p(\vec{Y})^{\beta}}_{\text{ヘッド部}} \text{ :- } \underbrace{Y_1 \overset{\beta_1}{=} t, \cdots}_{\text{ガード部}} \mid \underbrace{b_1, b_2, \cdots}_{\text{ボディ部}}.$$

という形をしている. ここで $X, Y, \cdots$ は論理変数, $\alpha, \beta, \cdots$ は優先度変数, $\vec{X}$, $\vec{\alpha}$ は変数の並び, t は関数項を表す. ボディ部及びトップレベルのゴール $(b_1, b_2, \cdots)$ は, 述語呼出し $(p(\vec{X})^{\alpha})$, 能動単一化 $(X \overset{\alpha}{=} t)$, 優先度指定 $((\vec{\alpha}^i) \succ \xi^o \succ (\vec{\beta}^i))$, 組込み述語から構成される.

織田信長では, 具体化を行う出現を高々 1 回に制限するために, モード解析 [11] を適用する. この制限は, 優先度変数に最終的に具体化された値がその述語や能動単一化を実行した時点での優先度に等しいことを保証するためである. この制限を満たすプログラムは優先度に関して well-moded であると言う. well-moded でないプログラムは実行不可であり, 本論文では扱わない. 最大優先度 $\top$ は $\forall\alpha.\top \succ \alpha \vee \top = \alpha$ と定義される.

優先度指定の一般形は $(\vec{\alpha}^i) \succ \xi^o \succ (\vec{\beta}^i)$ である. これは, すでに具体化された $\vec{\alpha}$, $\vec{\beta}$ という複数の優先度変数から, この大小関係を満たす新たな 1 つの優先度 ξ を具体化すると解釈する[4] (out モードの出現が 1 個所しかない点に注意). $\top \succ \xi^o \succ ()$ は, 既に存在している他の優先度とは全く関係を持たない新しい優先度変数を具体化することを意味する. 本稿のプログラムでは, 例えば $() \succ \alpha^o \succ (\beta^i)$, $(\alpha^i, \beta^i) \succ \gamma^o \succ ()$, $\top \succ \xi^o \succ ()$ に対して $\alpha^o \succ \beta^i$, $(\alpha^i, \beta^i) \succ \gamma^o$, $\top \succ \xi^o$ という省略記法を用いる.

優先度指定においてのみ優先度変数のモードを明示するのは, 通常の CLL プログラムにおいて, 算術演算子の各引数が出現文脈に依存せず, 暗黙的に in/out モードを持つのと同様である[5]. 意味のある well-moded な優先度付けに制限するため, $\succ$ に関して推移律は成立するが, 反射律と反対称律は成立しないとする [4]. どの述語呼出しや能動単一化を次に実行すべきかを決定する時, 一般には複数個の優先度が関係する. この複数個の優先度を**集合優先度 (aggregate priority)** と呼ぶ. 集合優先度の定義や比較は後述する.

述語呼出し $p(\vec{X})^{\alpha}$ や能動単一化 $X \overset{\alpha}{=} t$ における α という出現は in 出現であり, ここにどこか別の out 出現で具体化された値が渡って来る. 定義節側ヘッド部及びガード部 $p(\vec{Y})^{\beta}$:- $Y_1 \overset{\beta_1}{=} t, \cdots \mid \cdots$ における $\beta, \beta_1, \cdots$ は out 出現であり, これにより $\beta, \beta_1, \cdots$ が具体化される.

ボディ部の実行は FGHC の実行に準じて次の 3 ステップから成る.

ステップ 1: ボディ部 $b_1, b_2, \cdots$ から実行可能なものを選ぶ.

能動単一化は, 優先度変数が具体化されているなら実行可能. 優先度指定は, in モードの優先度変数が全て具体化されているなら実行可能. 組込み述語は, 引数変数が十分具体化されていれば実行可能. 述語呼出し $p(\vec{X})^{\alpha}$ は, α が具体化されており, 述語名と arity の一致するヘッドを持つ定義節が存在し, その定義節に $\{\vec{X}/\vec{Y}\ \alpha/\beta\}$ という置換を適用し, ガード部が成功する時に実行可能 (一般にそのような定義節は複数個ある).

ステップ 2: 実行可能な各 b_i の優先度 (集合優先度) を計算する.

能動単一化 $X \overset{\alpha}{=} t$ の集合優先度は $\{\alpha\}$ であり, 優先度指定と組込み述語の集合優先度は $\{\top\}$ である. ステップ 1 で, 述語呼出し $p(\vec{X})^{\alpha}$ に $p(\vec{Y})^{\beta}$:- $Y_1 \overset{\beta_1}{=} t, \cdots \mid b_1, b_2 \cdots$. という定義節が対応する場合, 集合優先度は $\{\alpha(=\beta), \beta_1, \cdots\}$ である. 非決定的な述語の場合は, 実行可能な定義節ごとに集合優先度が計算される. ガード部の受動単一化 $Y_1 \overset{\beta_1}{=} t$ に付与されたモード付き優先度変数 $(\beta_1, \cdots)$ の優先度は次のように計算する. $Y_1 \overset{\beta_1}{=} t$ に値を供給する能動単一化の連鎖を $X_1 \overset{\alpha_1}{=} X_2 \wedge X_2 \overset{\alpha_2}{=}$

[4]モードの記法について: 肩に ^{i} がついた変数を in (読出し) モードの出現, ^{o} がついた変数を out (書込み) モードの出現と呼ぶ. 直観的には, 変数は out モード出現によって値が具体化され, in モード出現によってその値が読み出されると考えればよい.

[5]例えば, X is Y + 1 という演算は, Y が具体値を持って初めて計算可能となる. この時, 暗黙の内に X^o is $Y^i + 1$ というモードが仮定されている.

$X_3 \wedge \cdots \wedge X_n \stackrel{\alpha_n}{=} t$ とする．この時 β_1 には $\alpha_1 \cdots \alpha_n$ の中で最小の優先度 $\alpha_1 \downarrow \cdots \downarrow \alpha_n$ が具体化される．ここで，

$$\alpha_i \downarrow \alpha_j = \begin{cases} \alpha_j & \cdots \alpha_i = \alpha_j \text{ あるいは } \alpha_i \succ \alpha_j \text{ の場合} \\ \alpha_i \succ \beta \wedge \alpha_j \succ \beta \text{ をみたす fresh な } \beta & \cdots \text{ それ以外の場合} \end{cases}$$

$\downarrow$ は可換である．また，$X \stackrel{\alpha}{=} f(Y_1, Y_2, \cdots) \wedge X \stackrel{\beta}{=} f(Z_1, Z_2, \cdots)$ からは，$Y_i \stackrel{\top}{=} Z_i$ $(i = 1, 2, \cdots)$ が演繹される．優先度変数を付与した等式理論の詳細は別稿に譲る．

ステップ 3: 極大な集合優先度を持つ b_i を任意に 1 つ選び実行する (committed choice).

ある集合優先度 π が集合優先度の集合 Π において極大であるとは，$\forall \phi \in \Pi. \neg(\phi \succ \pi)$ と定義される．さらに $\{\vec{\alpha}\}$ と $\{\vec{\beta}\}$ を空でない優先度変数の集合，σ を優先度指定の集合とする．この時，σ に関する $\{\vec{\alpha}\}$ と $\{\vec{\beta}\}$ 間の大小関係は次のように定義する．

$$\sigma \models \{\vec{\alpha}\} \succ \{\vec{\beta}\} \quad \text{iff} \quad \{\vec{\alpha}\} \neq \{\vec{\beta}\} \wedge (\forall \xi \in \{\vec{\alpha}\} \exists \eta \in \{\vec{\beta}\}. \sigma \models \xi \succ \eta \vee \sigma \models \xi = \eta) \wedge (\forall \eta \in \{\vec{\beta}\} \exists \xi \in \{\vec{\alpha}\}. \sigma \models \xi \succ \eta \vee \sigma \models \xi = \eta).$$

例えば，α, β を $\alpha \succ \beta$ なる優先度変数と仮定すると，$\{\alpha\} \succ \{\alpha, \beta\} \succ \{\beta\}$ が成り立つ．$\{\top\}$ という集合優先度は常に極大である．この優先度の集合に対して定義した $\succ$ も，優先度に対して定義した $\succ$ 同様に，推移律のみ成立し反射律と反対称律は成立しない．述語呼出し $p(\vec{X})^\alpha$ を実行する際は FGHC の実行と同じく，対応する定義節に $\{\vec{X}/\vec{Y}\ \alpha/\beta\}$ という置換を適用し commit する．

4 議論

織田信長に導入された優先度の特徴を以下にまとめる．

- 述語呼出しと能動単一化の優先度は全て呼出し側で付与する．
- 優先度変数に関して well-moded であり，優先度が具体化されて初めて実行可能となる．
- 優先度間の相対的な大小関係だけを指定する．

これらの特徴が第 2 節で述べた設計方針とどのように関連しているかを述べる．

述語呼出しの制御と節選択の制御: 優先度を取り込んだ CLL としてここでは KL1 を取り上げ，述語呼出しや節選択が優先度によってどのように制御されるかを織田信長と比較する．KL1 では，`@priority` でプロセスの優先度を，`alternatively` で節選択の優先度を制御している．`alternatively` の上下両側の節が commit 可能な時は，必ず上側の節が優先的に commit する．KL1 のプロセスに与えられる優先度は自然数で全順序が付いており，その値は動的に決められる．一方 `alternatively` はプログラマが静的に指定する．図 1 の KL1 プログラムにおいて，`Np` は述語呼出し `p(X,Y)` に付与された優先度であり，呼出し側

```
:- (X = a)@priority(10),
   (Y = b)@priority(20),
   p(X,Y)@priority(Np).

p(X,Y) :- X = a | true.
alternatively.
p(X,Y) :- Y = b | true.
```

図 1: KL1 における矛盾した優先度付け

$$:- \top \succ \alpha^o, \top \succ \beta^o, \gamma^o \succ \beta^i, X \stackrel{\beta}{=} s, Y \stackrel{\gamma}{=} t, r(X,Y)^\alpha.$$

$$r(X,Y)^\alpha :- X \stackrel{\beta}{=} s \mid \vec{b}_1.$$
$$r(X,Y)^\alpha :- Y \stackrel{\gamma}{=} t \mid \vec{b}_2.$$

図 2: 織田信長における節選択の制御

が適当な自然数を具体化しているとする．この時 `p(X,Y)` の commit は `Np` の値の影響を受ける．つまり，`Np` > 10 の場合は第 1 定義節が，`Np` ≤ 10 の場合は第 2 定義節が選択される．この振る舞いは，`X = a` と `Y = b` に動的に与えられた優先度の大小関係と，`alternatively` によって `p/2` の定義節間に静的に与えられた優先関係が食い違っていることに起因する．このように，KL1 における矛盾した優先度付けは，結果的に述語呼出し制御と節選択制御の干渉を引き起こす．

次に同様の 織田信長プログラムを図 2 に示すが, ここでは KL1 のような干渉は生じない. プログラム中, α と γ の間に大小関係が指定されていないので, $r(X,Y)^\alpha$ と $Y \stackrel{\gamma}{=} t$ のどちらが先に実行されるかは非決定的である. まず, $r(X,Y)^\alpha$ を先に実行しようとすると $Y \stackrel{\gamma}{=} t$ が実行されるまで suspend する. その後 $\vec{b}_2$ が実行される. もし $r(X,Y)^\alpha$ が $X \stackrel{\beta}{=} s$ や $Y \stackrel{\gamma}{=} t$ よりも後に実行されたとしても, r の第 1 定義節, 第 2 定義節の集合優先度は各々 $\{\alpha,\beta\}$, $\{\alpha,\gamma\}$ なので, α の優先度にかかわらず常に $\vec{b}_2$ の節が選択される. 従って r の節選択の制御は β と γ の大小関係によってのみ決まり, 述語呼出しに付与された α の値は影響しない. 織田信長では, 述語呼出しの優先度と節選択の優先度を動的に矛盾なく指定できる.

優先度の具体化: 先に, 我々は優先度の持つ論理的な意味に興味があると述べた. 実行中に述語や能動単一化の呼出し順序を決めるために優先度の大小関係が参照されるが, 優先度に論理的な意味を持たせるためには, この大小関係が実行過程を通じて不変でなければならない. つまり, 優先度変数が十分具体化されるまで, その優先度が付加された述語や能動単一化の呼出しを遅らせる必要がある. 織田信長はこの要請を満たすために, 優先度変数への書込みを高々 1 回に制限した.

ここで, 優先度変数への書込みが複数回ある well-moded でないプログラムを考えてみよう (図 3). 優

$$
\begin{array}{ll}
\text{:- } \top \succ \alpha^o, \alpha^i \succ \beta^o, \alpha^i \succ \gamma^o, & \\
\quad r(X,Y,\beta,\gamma)^\alpha, & r(X,Y,\_,\gamma)^\alpha \text{ :- } X \stackrel{\beta}{=} a \ \mid \gamma^o \succ \beta^i. \\
\quad X \stackrel{\beta}{=} a, Y \stackrel{\gamma}{=} b. & r(X,Y,\beta,\_)^\alpha \text{ :- } Y \stackrel{\gamma}{=} b \ \mid \beta^o \succ \gamma^i.
\end{array}
$$

図 3: 優先度変数へ複数回の書込みがあるプログラム

先度変数 β, γ に対して, トップゴールの $\alpha \succ \beta, \alpha \succ \gamma$ と $r/4$ の定義節のボディ部の 2 個所で制約がかけられている. $r/4$ のヘッド部の _ は void 変数である. まず $r(X,Y,\beta,\gamma)^\alpha$ が実行され suspend する. 次に β と γ の間には大小関係が指定されていないので, 非決定的に例えば $X \stackrel{\beta}{=} a$ が先に実行されるとしよう. すると suspend していた $r/4$ の第 1 定義節が resume し $\gamma \succ \beta$ という制約が課される. しかし, この時点で β を付与した能動単一化は実行済みであるのに対し, γ を付与した能動単一化は未実行である. 逆に $Y \stackrel{\gamma}{=} b$ を先に実行した場合も同様である. このように, 図 3 のプログラムでは, 最終的に得られる優先度間の大小関係の中に, 実行順序に関する情報は殆んど反映されない.

織田信長では, 最終的に優先度変数に具体化された値が呼出し時の優先度に等しいので, 後述するような優先度グラフ (第 6 章) から実行に関して意味のある情報を抽出できる.

直感的かつ必要最小限の表現: 織田信長での $\beta^i \succ \alpha^o \succ \gamma^i$ といった優先度指定の方法は, 自然数による指定よりも直感的である. さらに重要なのは, プロセスの間に関係が付けられない, 付けけたくないという場合である. 織田信長の場合, 例えば, 優先度 α, β, γ に関して, $\alpha \succ \beta$ 及び $\gamma \succ \beta$ という大小関係を付けただけでは, α と γ の間に大小関係は付かない. これは例えば, 分散環境の異なるプロセッサ上での処理をモデル化することができる. 従来の自然数による優先度の指定では, 優先度的に無関係にあるプロセスを抜き出したりすることはできない.

5 プログラム例

イベントループ: 織田信長の節選択機能と KL1 の `alternatively` 機能の対応を見るために, イベントループによる例外処理 (図 4, 5) を考える.

図 5 の述語呼出し $loop(Sig,D)^\alpha$ において, Sig 上のメッセージを D 上のメッセージより優先的に受信するには, Sig を受信する節を commit に関してより優先する必要がある. $loop$ の第 1 定義節の集合優先度は $\{\alpha,\beta\}$, 第 2 定義節の集合優先度は $\{\alpha\}$ であり, さらに $\beta \succ \alpha$ なので, Sig と D 共にメッセージが届いていれば常に Sig の方が優先される.

マージャ: 外部で非同期に生じるイベントを受信するようなプロセスを CLL で記述するには, 前述のイベントループか, ここで取り上げるマージャを用いなければならない. プロセス $merge$ は, $interrupt$ からのメッセージを $mandatory$ からのメッセージより優先して受信したいとしよう. 織田信長において, $merge(Sig,D,Z)$ が受信する Sig 上のメッセージを D 上のメッセージに優先させる方法は 2 つある.

```
main :- current_priority(A) |
  B := A + 3,
  interrupt(Sig)@priority(B),
  loop(Sig,init).

interrupt(Sig) :- current_priority(B),
                  exception(Ex)       |
  Sig = [(Ex,B)|T],
  interrupt(T).

loop(Sig,D) :- Sig = [(Ex,B)|T]    |
  handler(Ex)@priority(B),
  loop(T,D).
alternatively.
loop(Sig,D) :- current_priority(A) |
  body(D,R),
  G := A - 1,
  loop(Sig,R)@priority(G).
```

図 4: KL1 によるイベントループ

$:- \top \succ \alpha^o,$
$\quad \beta^o \succ \alpha^i,$
$\quad interrupt(Sig)^\beta,$
$\quad loop(Sig, D)^\alpha, D \stackrel{\alpha}{=} init.$

$interrupt(Sig)^\beta$:- $exception(Ex)$ |
$\quad Sig \stackrel{\beta}{=} [Ex|T],$
$\quad interrupt(T)^\beta.$

$loop(Sig, D)^\alpha$:- $Sig \stackrel{\beta}{=} [Ex|T]$ |
$\quad handler(Ex)^\beta,$
$\quad loop(T, D)^\alpha.$
$loop(Sig, D)^\alpha$:- $true$ |
$\quad body(D, R)^\alpha,$
$\quad \alpha^i \succ \gamma^o,$
$\quad loop(Sig, R)^\gamma.$

図 5: 織田信長によるイベントループ

$:- \top \succ \alpha^o, \beta^o \succ \alpha^i,$
$\quad \top \succ \gamma^o,$
$\quad merge(Sig, D, Z)^\gamma,$
$\quad interrupt(Sig)^\beta,$
$\quad mandatory(D)^\alpha.$

$merge(X, Y, Z)^\gamma$:- $X \stackrel{\beta}{=} [H|Xs] \mid Z \stackrel{\beta}{=} [H|Zs], merge(Xs, Y, Zs)^\gamma.$
$merge(X, Y, Z)^\gamma$:- $Y \stackrel{\alpha}{=} [H|Ys] \mid Z \stackrel{\alpha}{=} [H|Zs], merge(X, Ys, Zs)^\gamma.$

$interrupt(Sig)^\beta$:- 図 5 に同じ

$mandatory(D)^\alpha$:- $true$ |
$\quad gen\_data(X)^\alpha,$
$\quad D \stackrel{\alpha}{=} [X|Ds], \alpha^i \succ \gamma^o,$
$\quad mandatory(Ds)^\gamma.$

図 6: 織田信長によるマージャ

(1) メッセージに付加する優先度で制御する; メッセージに付加する優先度をその送信プロセスの優先度と同じくし, プロセスの優先度を $interrupt \succ mandatory$ の順にする. $merge$ の優先度は何でも構わない. (2) プロセスの優先度を制御する; メッセージに付加する優先度は全て同じにして, プロセスの優先度を $interrupt \succ merge \succ mandatory$ の順にする. (2) の方法なら KL1 でも実現できるが, (1) の方法は 織田信長でしか実現できない. ここでは (1) の方法に基づくマージャのプログラムを図 6 に示す. $merge$ の集合優先度は定義節の各々について $\{\gamma, \beta\}$, $\{\gamma, \alpha\}$ である. $\beta \succ \alpha$ であることから, γ が α や β とどのような大小関係にあっても $\{\gamma, \beta\} \succ \{\gamma, \alpha\}$ が成り立つ. また $merge$ の定義から, 受信したメッセージの優先度を保持したまま, さらに後段のプロセスにメッセージを送信することが分かる.

2 人の phil と 1 本のフォーク: この問題は, 2 つの phil というプロセスが 1 本のフォークを取り合って非決定的かつ排他的に eat するというものである. 図 7 の KL1 プログラムは, 2 つの `phil` が規則正しく交互に `eat` する[6]. ここで一方の `phil/1` の `(1)`, `(2)` の部分を例えば `H := T - 10`, `TN := E - 10` と変更すると (`phil10` と呼ぶ), `phil` が 7 回 `eat` する間に `phil10` が 1 回 `eat` するようになる. これは, 優先度が自然数のため, 2 つの `phil` 間に (プログラマの意図しない) 順序が付いてしまうからであり, プログラム全体の動作を理解するには 2 つの `phil` に付加された優先度の相互関係まで考慮に入れなければならない.

これに対し, 図 8 の 織田信長のプログラムでは, 明示的に順序を指定しない限り順序が付かないので 2 つの $phil$ の動作は非決定的かつ並行に進む. 織田信長では $phil$ どうしの優先度の大小関係を逐次的に追跡せずとも, プログラム全体の動作を理解できる.

[6] このように動作するためには, 逐次処理系等の条件も多少必要ではあるが, 多くのプログラマはこのような動作は意図していないだろう.

```
:- P := 10000,
   phil(M1)@priority(P),
   phil(M2)@priority(P),
   mutex(M1,M2)@priority(P).

phil(M) :- current_priority(T) |
   H := T - 1,                              (1)
   think@priority(T),
   lock_fork(M,E,Ms)@priority(H),
   TN := E - 1,                             (2)
   eat@priority(E),
   phil(Ms)@priority(TN).

lock_fork(M,E,Ms) :- current_priority(H) |
   M = [E,H|Ms].

mutex(M1,M2) :- M1 = [E,H|M1s],
                current_priority(P) |
   min(P,H,Min),
   E := Min - 1,
   F := E - 1,
   mutex(M1s,M2)@priority(F).
mutex(M1,M2) :- M2 = [E,H|M2s],
                current_priority(P) |
   min(P,H,Min),
   E := Min - 1,
   F := E - 1,
   mutex(M1,M2s)@priority(F).

min(P,H,M) :- P > H  | M := H.
min(P,H,M) :- P =< H | M := P.
```

図 7: KL1 による 2 人の phil と 1 本のフォーク

$$
\begin{aligned}
&\text{:-}\ \top \succ \alpha^o,\\
&\quad phil(M_1)^\alpha,\\
&\quad phil(M_2)^\alpha,\\
&\quad mutex(M_1, M_2)^\alpha.
\end{aligned}
$$

$$
\begin{aligned}
&phil(M)^\beta \text{ :- } true \mid\\
&\quad \beta^i \succ \gamma^o,\\
&\quad think^\beta,\\
&\quad M \stackrel{\gamma}{=} [\varepsilon | M_s],\\
&\quad \varepsilon^i \succ \delta^o,\\
&\quad eat^\varepsilon,\\
&\quad phil(M_s)^\delta.
\end{aligned}
$$

$$
\begin{aligned}
&mutex(M_1, M_2)^\alpha \text{ :- } M_1 \stackrel{\gamma}{=} [\varepsilon | M_{1s}] \mid\\
&\quad (\gamma^i, \alpha^i) \succ \varepsilon^o,\\
&\quad \varepsilon^i \succ \eta^o,\\
&\quad mutex(M_{1s}, M_2)^\eta.\\
&mutex(M_1, M_2)^\alpha \text{ :- } M_2 \stackrel{\gamma}{=} [\varepsilon | M_{2s}] \mid\\
&\quad (\gamma^i, \alpha^i) \succ \varepsilon^o,\\
&\quad \varepsilon^i \succ \eta^o,\\
&\quad mutex(M_1, M_{2s})^\eta.
\end{aligned}
$$

図 8: 織田信長による 2 人の phil と 1 本のフォーク

6 まとめに代えて

これまで示した 織田信長のプログラム例は, 宣言的プログラミングの枠組において, 優先度が実行に関するプログラマの意図を適切に表現する手段に成り得る可能性を示唆している.

- 関連研究
  制約に優先度を付加する研究では, 例えば CLP/P という言語が提案されている [5]. CLP/P では, 各制約に優先度アトムを付加し, 優先度アトム間の論理的 implication で優先度の順序付けを表現する. この論理的 implication と織田信長の優先度指定 $\succ$ の間に対応関係を見いだすことができる. 実行機構の観点からは, CLP/P はバックトラックしながら高い優先度の制約がより多く充足されるように解を探していくのに対し, 織田信長では並行実行を優先度を用いて制御している. 言語の意味モデルの観点からは, CLP/P の最適な解ではより高い優先度が付加された制約は必ず満たされているのに対し, 織田信長では最高優先度の述語でも実行可能でない限りは実行されないので, モデルに含まれるとは限らない. 織田信長のモデル論的な意味を構築する際は, CLP/P の意味論が参考になるであろう.

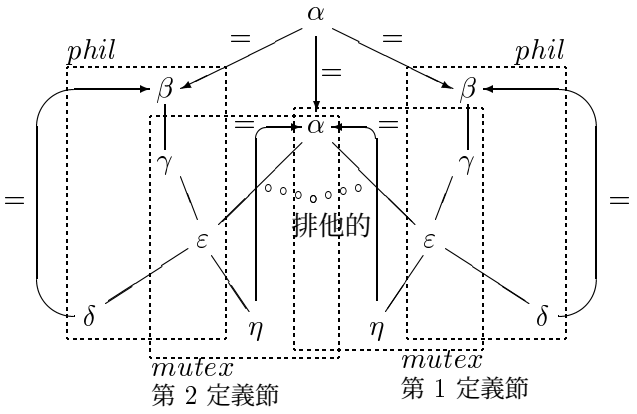
我々は, プログラム中の優先度を静的に解析することで実行に関する以下のような性質が検出できると考えている.

- スレッドの抽出, ゴールスケジューリングのための情報
  優先度グラフの向きと, 論理変数のモードの向きの整合性を検査することで, 実行中断のないスレッドの抽出や効率の良いゴールスケジューリングが可能となる. 逆に, 互いに優先度の関係がないスレッドを抽出できれば, それらスレッドを分散実行の単位と見なすことができる.

- 優先度の検査を省略できる場合の検出
  織田信長の実行モデルを単純に実装すると, 効率上次のような問題が生じる.
  - リダクション毎に優先度グラフから極大な優先度を持つゴールを決定しなければならない.
  - 単一化の連鎖とその優先度を保持し, 上と同様の計算をしなければならない.

  しかし, 優先度の実行時計算を省略できる幾つかの実行パターンがわかっている. そのようなパターンを解析によってできるだけ抽出することにより, 従来の CLL 並みの実行効率が期待される. 優先度がないと記述できないプログラムが存在する以上, このコストは見合うものであると考える.

- 排他性などに関する性質
  例えば図 8 のプログラム (2 人の哲学者) から優先度の大小関係を抽出すると, 右図のような優先度グラフが得られる. 図中, 優先度間の線分は $\succ$ 関係を, $=$ の付記された矢印は (再帰) 呼出しにより単一化された優先度変数の関係を表現している. このグラフより, プロセス *eat* は排他的に実行されることが分かる. 何故なら, どんな優先度も必ず ε に対して順序が付き, ε という優先度を持つプロセスは 1 個 (*eat* のみ) だからである.



さらに優先度の静的解析の応用として, 優先度を含むプログラムの変換, 実時間プロセススケジューリング等も考えている.

最後に, 我々が FGHC を基本として, それに優先度を付与し 織田信長を設計した動機の 1 つは, 優先度の付与が意味論にどのような影響を与えるのかを調べるためであった. 織田信長の意味論を構築することで, 優先度という概念を形式化しその本質的な理解を目指そうと思う.

謝辞 上田和紀教授 (早稲田大学), 近山隆教授 (東京大学) を初めとする KLIC Task Group (AITEC) の皆様からは討論を通じて大変有意義な数多くのコメントを頂戴しました. SPA'99 査読者の方々からも貴重なコメントを頂戴しました. 石井健一郎部長 (旧 NTT 基礎研究所 情報科学研究部) には常日頃より暖かい励ましを頂戴しました.

参考文献

[1] Chow, R., and Johnson, T., *Distributed Operating Systems and Algorithms*, Addison Wesley, 1997.
[2] Fidge, C. J., A Formal Definition of Priority in CSP, *ACM Trans. on Prog. Lang. and Sys.*, Vol.15, No.4, Sep. 1993, pp.681–705.
[3] Gregory, S., Experiments with Speculative Parallelism in Parlog, In *Proc. of ISLP'93*, 1993.
[4] Gregory, S., and Ramirez, R., Tempo: a declarative concurrent programming language, In *Proc. of the 12th ICLP*, 1995.
[5] 服部隆志, 優先度付き制約論理型プログラミング, コンピュータソフトウェア, Vol.9, No.6 (1992), pp.48–57.
[6] Huntback, M., Speculative Computation and Priorities in Concurrent Logic Languages, In *Proc. of ALPUK'91*, 1991.
[7] Huntback, M. and Ringwood, G., Programming in Concurrent Logic Language, *IEEE Software*, Nov. 1995.
[8] Lloyd, J. W., *Foundations of Logic Programming*, Second, Extended Edition, Springer-Verlag, 1987.
[9] Murakami, M., A Declarative Semantics of Flat Guarded Horn Clauses for Programs with Perpetual Processes, *Theoretical Computer Scienece* 75 (1990) 67–83, North-Holland.
[10] Ueda, K. and Chikayama, T., Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, Vol.33, No.6 (1990), pp.494–500.
[11] Ueda, K. and Morita, M., Moded Flat GHC and Its Message-Oriented Implementation Technique, *New Generation Computing*, Vol.13, No.1 (1994), pp.3–43.