マイクロタスク型クラウドソーシング処理の変換

権守　健嗣† 　森嶋　厚行††,††† 　北川　博之††††

† 筑波大学大学院 システム情報工学研究科　〒 305–8550 茨城県つくば市春日 1–2
†† 筑波大学 図書館情報メディア系/知的コミュニティ基盤研究センター　〒 305–8550 茨城県つくば市春日 1–2
††† JST さきがけ
†††† 筑波大学 システム情報系　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †s1320687@u.tsukuba.ac.jp ††mori@slis.tsukuba.ac.jp ††††kitagawa@cs.tsukuba.ac.jp

あらまし　近年，Amazon Mechanical Turk 等のマイクロタスク型クラウドソーシングプラットフォームの登場により，それらを利用した問題解決が注目を集めている．本論文では，そのようなマイクロタスク型クラウドソーシング処理の最適化について議論する．具体的には，与えられたマイクロタスクの集合を，同等の結果が期待される他のマイクロタスクの集合に変換する手法を提案する．本提案手法の特徴は次の通りである．(1) 既に存在するマイクロタスクからの書き換えを可能にする．(2) 既存のリレーショナル代数演算式等の変換とは異なる変換が可能になる．本論文では，変換手法の説明に加え，提案手法を利用することにより様々な観点からの最適化を行えることを示す．

キーワード　クラウドソーシング，変換，最適化

1. はじめに

近年，Amazon Mechanical Turk [1]（以下，MTurk）等のマイクロタスク型クラウドソーシングプラットフォームの登場により，それらを利用した問題解決が注目を集めている．**マイクロタスク**とは，クラウドソーシングする作業（タスク）の中でも特に処理に必要な時間が短いタスクのことである．例えば，人に画像を見せ，そこに猫が写っているかどうかを判定するタスクはマイクロタスクであるといえる．MTurk などのプラットフォームは，マイクロタスクを依頼したい人 (リクエスタ) がタスクを登録するためのタスクプールと，タスクを処理する人 (ワーカ) が登録されているマイクロタスクを処理するための仕組みを提供している．

これらのプラットフォームのいくつかは，外部ソフトウェアから呼び出し可能な API を提供しているため，その API を呼び出すソフトウェアを開発することにより，複数のマイクロタスクを組み合わせた複雑なクラウドソーシングを実現することができる．例えば，マイクロタスクを組み合わせてのソート [6] や，SQL 処理の一部 [2] [7] が行われている．

一般に，ある問題をマイクロタスクを用いて解決したいときには，その実装方法は様々なものが存在する．本論文では，ある問題解決を実装するためにデザインされた，一般に複数のマイクロタスクと機械的処理の組み合わせを**実行プラン**と呼ぶ．例えば，風景画像の集合が与えられたとき，これらの画像を美しさで順序付けを行いたい場合には，少なくとも次のような二つの実行プランが考えられる．

（1） ワーカに各風景画像に美しさを 1 から 100 の値で付与するタスクを行ってもらい，そのスコアを用いてソートを行う．

（2） ワーカに二つの風景画像を見せてどちらが美しいか判定してもらい，その順序関係を用いてソートを行う．

以上の二つの実行プランどちらにおいても，結果として人々が美しいと考える順に並べられた写真の列が得られることが期待できる．しかし，上に示した二つの実行プランでは，実際にワーカに行ってもらうマイクロタスクの数が大きく異なる．また，タスクの処理に必要な入力インターフェースも異なる．すなわち，前者ではキー入力が必要であり，後者では，二択のボタンで可能である．

クラウドソーシングにとって，タスクの入力インターフェースは重要な要素である．その理由は，タスクの処理環境によっては，タスク処理のための入力インターフェースが限定される場合があるからである．例えば，タスクをクラウドソーシングプラットフォーム上で行うだけでなく日常環境に埋め込む試みとして，自動販売機上でのタスクの回答 [8] や床を歩くことによるタスクの回答 [4] などが行われているが，これらの環境では，フルキーボードを必要としないテンキーでの入力や選択式などで回答できるタスクである必要がある．

本論文では，マイクロタスク定義を含む実行プランを入力とし，それと同等の結果が期待される他の実行プランに変換する手法を提案する．本論文の貢献は次の二つである．

(1) マイクロタスクの書き換えによる最適化

我々の知る限り，本提案手法は，既存のマイクロタスクから別のマイクロタスクへの変換を試みる初めての手法である．既存手法でもクラウドソーシングの最適化の議論はあったが [2] [6] [7]，それらは，より抽象度の高い記法 (SQL 等) を記述し，それをマイクロタスクで実行するための実行プランを生成するモデルで行われていた．しかし，マイクロタスク型クラウドソーシングプラットフォームにおいてはマイクロタスク定義を登録することが一般的であり，それらを元に最適化などの目的でタスク変換を行うことは大きな需要があると考えられる．提案手法では，一般的にマイクロタスク型クラウドソーシングプラットフォームで用いられるマイクロタスク定義を一般化した CTS

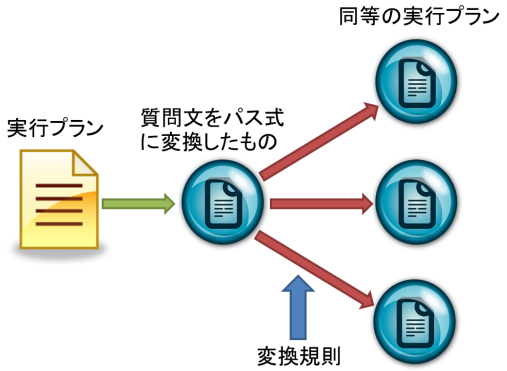

図 1　マイクロタスクの書き換えプロセス

ルール [3] (3 節) によって記述された実行プランの書き換えを行う．

マイクロタスクの書き換えが困難である理由の一つは，ワーカへの質問文 (「この国の公用語はスペイン語ですか？」等) が自然言語であることである．本手法のアイデアは，その自然言語をパス式に置き換えるというプロセスを挟むことである（図 1）．例えば，先の質問は `country[?language=spanish]` と書き換える．この書き換えは現在人手を要するが，局所的な変更以外は不要であるため，与えられたマイクロタスクの定義を含む実行プランをリバースエンジニアリングして SQL などのより抽象度の高い記法に変更するよりも容易であることが期待できる．パス式によるワーカへの質問文の記述については 4 節で記述する．

(2) 新規性のある変換規則

本論文の 5 節では，CTS ルールの集合として書かれた実行プランを変換するための規則を導入する．マイクロタスク型クラウドソーシングの実行プランにおいては，計算機の処理だけでなくマイクロタスクを通じた人による処理も含まれている．したがって，これらの変換規則は，既知の変換規則 (リレーショナル代数演算式等の変換) と等価な変換規則だけでなく，これまで議論されていなかった規則も含まれている．

新規性のある規則の第一の例は，人の情報源固有の変換規則である．例えば，公用語がスペイン語であるような国の一覧を作りたいとする．このとき，実行プランの一つは，国名一覧を作成した後，先のタスク「この国の公用語はスペイン語ですか？(`country[?language=spanish]`)」を 実行するものであるが，この実行プランを，直接答えを入力させるタスク「公用語がスペイン語である国はどこですか？(`country[language=spanish][name=?]`)」に変換することは，リレーショナル代数演算式の変換規則では導出できない．

第二の例は，多様なマイクロタスク実行環境を考慮したマイクロタスクの書き換えである．例えば，図 2 のように，キーボードを使った入力が必要となるタスクを，タッチによる入力で十分なタスクに変換すると，PC 上以外の環境で処理が困難であったタスクが，タブレット上や [4] で示されているような床の上といった多様な環境で処理が容易となる．

本論文の構成は次の通りである．2 節では関連研究について

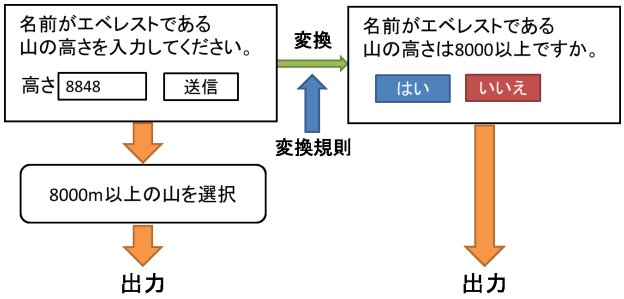

図 2　より単純なインターフェースへのマイクロタスクの書き換え例

述べる．3 節では，本論文で，マイクロタスクの記述形式として用いる CTS ルールについて説明し，4 節では CTS ルール中の質問文に相当する項目をパス式で記述する際の記述方法について説明する．5 節では，マイクロタスク集合間に成り立つ変換規則を示し，6 節でその変換規則についての議論を行う．7 節はまとめと今後の課題である．

2. 関連研究・関連システム

近年，クラウドソーシング処理について，多くの論文で議論されている．それらの議論には大きく分けて二つの対象がある．一つは特定の演算に関する処理である．もう一つは SQL 問合せの処理である．

(1) 特定の演算についての処理

これまで，データベースの分野では，伝統的なリレーショナルデータベース演算の処理をクラウドソーシングによって行うための研究が数多く行われてきた．例えば, [6] は，結合演算におけるタスク数の最適化を議論しており，Join Pre-filter を用いることで結合に必要な判定の数を減らすことを提案している．Join Pre-filter では結合の対象となるテーブルのレコードに対し，予め値が非両立となる属性を付与する．例えば，異なる画像集合同士を同じ人が写っているという条件で結合する際には，結合処理を始める前に，各画像について，写っている人の性別の入力をワーカに行ってもらう．この前処理によって，男性の画像集合同士と女性の画像集合同士で結合することが可能となり，必要な判定の数を削減できる．また，タスクの実装におけるバッチの大きさの違いがもたらすタスク数の変化についても述べている．

また, [6] は，ソート演算における実行プランの違いも議論している．具体的には，比較ベースの実行プランとレーティングベースの実行プランを比較しており，それらのタスク数，精度について議論している．

[5] は，カウント演算における新しい実行プランと，効率化について議論している．具体的には，従来のラベリングベースの実行プランと新しく提案したカウントベースの実行プランを比較している．例えば，人が写った 2000 枚の画像集合について，女性が写っている画像の枚数をカウントすることを考える．その場合には，従来のラベリングベースの実行プランでは，まず，各画像について性別をラベリングし，最後に，性別が女性であるとラベリングされた画像の枚数を機械的処理によって数

える．それに対し，提案したカウントベースの実行プランでは，画像集合からサンプリングした画像（例えば，50 枚）を同時にワーカに見せ，女性が写った画像が何枚あるかを推定してもらうことを何度か繰り返すことによって，画像集合に女性が写った画像が何枚あるかを推定する．[5] では，このような二つの実行プランについて，それぞれのタスク数，精度について議論している．

(2) SQL 問合せについての処理

もう一つの主要な研究は，SQL で書かれたクラウドソーシングの処理方式である．[7] では，特定の演算に限らず SQL 問合せ全体を通してのコストベースの最適化について議論している．与えられたクエリから実行可能なプランを探索し，金銭的コストが最も小さいプランを見つける．このときのコストは過去のクエリによって得たデータもしくは，元からデータベースに存在するデータから推定する．

既存の研究はいずれも与えられた抽象度の高いクエリから実行プラン (マイクロタスク処理を含む) を生成しているが，本研究では，既存の実行プラン (マイクロタスク処理) から異なる実行プランへの変換を議論するという点に新規性がある．

3. CTS ルール

本節では，本論文でマイクロタスクの記述形式として用いる CTS ルールの説明を行う．

図 3 のルール 0-2 は CTS ルールの例である．CTS ルールは Condition 部，Task 部，Store 部の三つ組から構成されるルールであり，リレーショナルデータベースに対してデータを読み書きするために用いる．一般に，一つの CTS ルールは，データベースから読みだしたデータを元にタスクを生成し，そのタスクの処理結果をデータベースに格納するという一連の処理を表す．ルール 0-2 では，データベース上の `Mountain` リレーションから各タプルの属性 `name` の値を読み出し，Task 部に記述された`$name` を各値に置き換えたタスクを生成する．そして，各タスクで入力された高さをデータベース上の `Mountain` リレーションの対応するタプルに格納する．

以下に CTS ルールの各要素について簡単に説明する．CTS ルールの詳しい記述方法は [3] にある．

Condition 部 タスクを生成するための条件を記述する．条件は，データベース上に存在する必要のあるタプルを零個以上の原子式の列として記述する．例えば，図 3 のルール 0-2 の Condition 部には，`Mountain(name)` という原子式が記述されており，`Mountain` リレーションに格納された山ごとにタスクを生成することを示している．また，図 3 のルール 0-1 のように Condition 部に原子式が一つも記述されていない場合には無条件にタスクを生成することを示している．

Task 部 タスクテンプレートを記述する．タスクテンプレートはタスクを行う上でワーカに提示される質問文や，タスクの結果の型などを記述したものである．質問文には，`$name` といった変数を用いることができる．変数の値は Condition 部でタプルごとに束縛されるため，タスクが生成される度に置き換わる．Task 部にはタスクテンプレートの項目として，Question

ルール 0-1

Condition	
Task	山の名前を入力してください
Store	`Mountain(name)`

ルール 0-2

Condition	`Mountain(name)`
Task	名前が`$name` である山の高さを入力して下さい
Store	`Mountain(name, height)`

ルール 0-3

Condition	`Mountain(name, height), height >= 8000`
Task	-
Store	`HighMountain(name)`

図 3　例．高さ 8000m 以上の山を集める実行プラン

（ワーカに提示する質問文を記述する項目）の他に Count（変数を同じ値で置き換えたタスクをいくつ生成するかを記述する項目）と Payoff（タスクを行ったことによりワーカが得られる報酬を規定する項目）を記述可能であるが，議論の簡単化のため本論文では省略し，Question の内容を直接記述する．

Store 部 タスクが処理されたことによって得られたデータをどのようにデータベースに格納するかを，一個以上の原子式の列として記述する．Store 部の各原子式の各項は変数もしくは定数から成り，変数の値は Condition 部で束縛されるか，もしくはタスク中で与えられる．

図 3 のルール 0-3 のように Task 部に記述がないルールでは計算機による計算を表現できる．ルール 0-3 では Condition 部には `Mountain` の原子式に加え，`height` が満たすべき条件が記述されており，それらの条件が満たされた場合に機械的に処理を行い，結果を格納することを示している．具体的には，`Mountain` に格納されたタプルの中から `height` が 8000 以上のものを `HighMountain` に挿入する処理を機械的に行うことを示している．

以上のように，CTS ルールはマイクロタスクと機械的処理を表現できる．全体として特定の目的の処理を行うための CTS ルールの集合を，本論文では**実行プラン**と呼ぶ．例えば，図 3 は，8000m 以上の山を列挙するための実行プランの一つである．

以下では，CTS ルールの集合を図 3 のような表の集合ではなく，$\{(C_1|T_1|S_1),(C_2|T_2|S_2),...\}$ といった組の要素を | で区切った三つ組の集合で表現する場合がある．この場合，各要素が空の場合には null と記述する．

4. パス式によるマイクロタスク記述

マイクロタスクの書き換えが困難である理由の一つは，ワーカへの質問文が自然言語で記述されていることである．そのため，本論文では，CTS ルールの Task 部の Question を自然文ではなくパス式を用いて記述することを提案する．図 4 は，図 3 の Task 部をパス式で記述したものである．

パス式を記述する際には，対象世界を記述した ER 図を仮定する．例えば，図 5 は，属性 `name` と `height` をもつ実体集合 `mountain` を記述した ER 図である．

図 4 中のパス式は，図 5 に示す対象世界を記述した ER 図を

ルール 1-1

Condition	
Task	`mountain[?name]`
Store	`Mountain(name)`

ルール 1-2

Condition	`Mountain(name)`
Task	`mountain[name=$name][?height]`
Store	`Mountain(name, height)`

ルール 1-3

Condition	`Mountain(name, height), height >= 8000`
Task	-
Store	`HighMountain(name)`

図 4　例．高さ 8000m 以上の山を集める実行プラン 1（パス式）

仮定して書かれている．例えば，図 4 のルール 1-2 では，属性 `name` が Condition 部で束縛された変数`$name` と一致する実体集合 `mountain` の属性 `height` の値を，人へ問い合わせることが記述されている．

パス式を書くための ER 図は，CTS ルールを記述する人が自由に決めて良い．CTS ルールの集合が与えられると，本手法では元となった ER 図を，CTS ルール中のパス式に基づき構築する．例えば，図 4 のパス式に記述されている実体，属性などの関係から，図 5 の ER 図を構築することができる．

パス式の構文は概要は次の通りである (詳細は表 1 に示す)．

実体の選択. 対象とする実体集合から実体を選択する際には，実体名に続いて “[ ]” 中に条件を記述する．例えば，図 4 のルール 1-2 では，`[name=$name]` という記述で，対象とする `mountain` が満たすべき条件を記述している．

人への問い合わせ. 人への問い合わせを記述する際には，選択された実体に続いて，“?” を接頭辞として入力を求める属性名や判定してもらう条件を “[ ]” 中に記述する．例えば，図 4 のルール 1-2 では，`[?height]` という記述で `mountain` の `height` の値を人へ問い合わせることを記述している．

変数の束縛. `$name` のように “$” を接頭辞とする記述は変数を表しており，Condition 部で束縛され，実際の質問文には具体的な値が代入される．例えば,「名前がエベレストである山の高さを入力してください」といった質問文のように，変数`$name` に「エベレスト」というような具体的な値が代入される．

実体間の関連. 実体間に成り立つ関連を表現する際には，“.” を用いる．例えば,「学生が所属する学部」を表現するには「`student.belong.faculty`」というパス式で，実体集合 `student` と `faculty` の間に `belong` という関連が成り立つことを記述する．また,「`student.?belong.faculty`」のように関連名の接頭辞に?を記述することで，その関連を挟んだ二つの実体集合間にその関連が成り立つかどうかを人に問い合わせることができる．

5. 実行プランの書き換え

本節では，本論文で提案する実行プランの書き換えについて説明する．まず，実行プランの書き換えについて例を用いて説明し，最後に，実行プランの変換規則を示す．

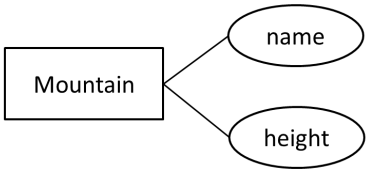

図 5　対象世界を記述した ER 図

ルール 2

Condition	
Task	`mountain[height>=8000][?name]`
Store	`HighMountain(name)`

図 6　例．高さ 8000m 以上の山を集める実行プラン 2

5.1 実行プランの書き換えの例

例として，図 4 に示した高さ 8000m 以上の山を集めるための実行プランを書き換える．変換の詳細は 5.2 節で説明するが，書き換えた実行プランの例を図 6 に示す．図 6 では，図 4 に示した実行プランと比較して，次のような違いがある．図 4 の実行プランでは，まず，山の名前を入力（ルール 1-1）し，次に入力された山の高さを入力（ルール 1-2）し，最後に 8000m 以上の山を選択していたのに対し，図 6 の実行プランは直接，高さが 8000m 以上の山の入力する．図 6 の実行プランは図 4 の実行プランと同様に，最終的に `HighMountain` に 8000m 以上の山が格納されるので同等な実行プランである．

5.2 実行プランの変換規則

本節では，実行プランの変換規則について説明する．実行プランの変換規則にはリレーショナル代数演算式の変換規則に相当する規則も存在するが，それらの規則については本論文では省略し，人からの入力などリレーショナル代数演算で処理できない処理を含む実行プランの書き換えについて説明する．

表 2 に現時点で把握している実行プランの変換規則を示す．表 2 中の θ は比較演算子であり，n は定数もしくは Condition 部で束縛された変数で既知の値である．

挿入・更新演算の変換規則

本論文中では人による値の入力を挿入演算，値の更新を更新演算と呼ぶ．

[規則 1: 値入力，判定 $\rightsquigarrow$ 値入力]

例）8000m 以上の山をワーカに問合せる実行プランとして，次の実行プランがあるとする．

```
{(null | mountain[?name] | Mountain(name)),
(Mountain(name) | mountain[name=$name][?height>=8000]
| HighMountain(name))}
```

これは，まず，山の名前を入力し，次に入力された山の高さが 8000m 以上かどうかを判定するという実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(null | mountain[height>=8000][?name] |
HighMountain(name))}
```

この実行プランは，高さ 8000m 以上の山を直接入力する実行プランである．表 2 の規則 1 はこの書き換えを一般化したもの

表 1　パス式の構成要素の一覧

表記	説明
実体集合名 [条件式]	条件式を満たす実体集合を表す．条件式は，属性名 θ 変数名もしくは属性名 θ 値という比較条件式，およびそれらの比較条件式に論理演算子 (and, or, not) を適用したもの．θ は比較演算子 (>，<，>=，<=，=，!=) を表す．
実体集合名 [?属性名:変数名]	?を接頭辞とする属性の値の入力を求め，指定された変数名の変数に値を結びつける．変数名を省略し，?属性名のみを記述した場合は属性名と同名の変数に値を結びつける．変数は Store 部で利用できる．
実体集合名 [?条件式]	条件式が満たされるかどうかの入力を求める．条件式は，属性名，変数名，値についての比較条件式，およびそれらの比較条件式に論理演算子（and, or, not）を適用したものである．
実体集合名.?関連名．実体集合名	?を接頭辞とする関連を挟んで隣り合う実体集合間にその関連が成り立つどうかの入力を求める．

表 2　変換規則一覧

規則 1(挿入) 値入力，判定 $\rightsquigarrow$ 値入力	$\{(\texttt{null} \mid x[?a] \mid X(a)), (X(a) \mid x[a = \$a][?b\theta n] \mid Y(a))\} \rightsquigarrow \{(\texttt{null} \mid x[b\theta n][?a] \mid Y(a))\}$
規則 2(挿入) 値入力，値入力，判定 $\rightsquigarrow$ 複数値入力	$\{(\texttt{null} \mid x[?a] \mid X(a)), (\texttt{null} \mid y[?b] \mid Y(b)), (X(a), Y(b) \mid x[a = \$a].?r.y[b = \$b] \mid R(a,b))\}$ $\rightsquigarrow \{(\texttt{null} \mid x[?a].r.y[?b] \mid R(a,b))\}$
規則 3(挿入) 値入力，値入力 $\rightsquigarrow$ 複数値入力	$\{(\texttt{null} \mid x[?a] \mid X(a)), (X[a] \mid x[a = \$a].r.y[?b] \mid R(a,b))\} \rightsquigarrow \{(\texttt{null} \mid x[?a].r.y[?b] \mid R(a,b))\}$
規則 4(更新) 値入力，更新 $\rightsquigarrow$ 複数値入力	$\{(\texttt{null} \mid x[?key] \mid X(key)), (X(key) \mid x[key = \$key][?a]) \mid X(key,a)\} \rightsquigarrow \{(\texttt{null} \mid x[?key,?a] \mid X(key,a))\}$
規則 5(更新) 値入力 $\rightsquigarrow$ 判定	$\{(X(a) \mid x[a = \$a][?b] \mid X(a,b))\} \rightsquigarrow \{(X(a), Y(a,b) \mid x[a = \$a][?b = \$b] \mid X(a,b))\}$
規則 6(選択) 未知値入力，条件計算 $\rightsquigarrow$ 判定	$\{(X(a) \mid x[a = \$a][?b] \mid X(a,b)), (X(a,b), b\theta n \mid \texttt{null} \mid Y(a))\} \rightsquigarrow \{(X(a) \mid x[a = \$a][?b\theta n] \mid Y(a))\}$
規則 7(選択) 具体値入力，条件計算 $\rightsquigarrow$ 判定	$\{(X(a) \mid x[a = \$a][?s] \mid X(a,s)), (X(a,s), s\theta n \mid \texttt{null} \mid Y(a))\} \rightsquigarrow \{(X(a) \mid x[a = \$a][?is\_s = true] \mid Y(a))\}$
規則 8(結合) 値入力，順序計算 $\rightsquigarrow$ 順序判定	$\{(X(a) \mid x[a = \$a][?s] \mid X(a,s)), (X(a:h, s:s1), X(a:l, s:s2), s1 > s2 \mid \texttt{null} \mid IsMoreS(h,l))\}$ $\rightsquigarrow \{(X(a:h), X(a:l) \mid x[a = \$h].?is\_more\_s.x[a = \$l] \mid IsMoreS(h,l))\}$
規則 9(結合) 値入力，一致計算 $\rightsquigarrow$ 一致判定	$\{(X(a) \mid x[a = \$a][?s] \mid X(a,s)), (X(a, s:s1), X(a:b, s:s2), s1 = s2 \mid \texttt{null} \mid IsSameS(a,b))\}$ $\rightsquigarrow \{(X(a), X(a:b) \mid x[a = \$a].?is\_same\_s.y[a = \$b] \mid IsSameS(a,b))\}$

である．

[規則 2: 値入力，値入力，判定 $\rightsquigarrow$ 複数値入力]

例）学生が学部に所属しているという関係をワーカに問合せる実行プランとして，次の実行プランがあるとする．

```
{(null | student[?name] | Student(name)),
(null | faculty[?name] | Faculty(name)),
(Student(name:s), Faculty(name:f) |
student[name=$s].?belong.faculty[name=$f] | Belong(s,f))}
```

これは，まず，学生の名前を入力し，次に，学部の名前を入力し，最後に，各学生が各学部に所属しているかどうかを判定するという実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(null | student[?name:s].belong.faculty[?name:f] |
Belong(s, f))}
```

この実行プランは，所属の関係にある学生と学部を直接入力する実行プランである．表 2 の規則 2 はこの書き換えを一般化したものである．

[規則 3: 値入力，値入力 $\rightsquigarrow$ 複数値入力]

規則 2 の例と同様のデータを求める実行プランとして次の実行プランも考えられる．

```
{(null | student[?name] | Student(name)),
(Student(name:s) |
student[name=$s].belong.faculty[?name:d] | Belong(s,d))}
```

この実行プランは，まず，学生の名前を入力し，次に入力された学生が所属する学部を入力するという実行プランである．この実行プランについても規則 2 の例で行った書き換えで得られた実行プランへ書き換えることができる．表 2 の規則 3 はこの書き換えを一般化したものである．

[規則 4: 値入力，更新 $\rightsquigarrow$ 複数値入力]

例）与えられた動物の名前から分類を追加してもらう更新演算を行う実行プランとして，次の実行プランがあるとする．

```
{(null | animal[?name] | Animal(name)),
(Animal(name) | animal[name=$name][?class] |
Animal(name, class))}
```

これは，まず，動物の名前を入力し，次に入力された動物の分類を入力するという実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(null | animal[?name, ?class] | Animal(name, class))}
```

この実行プランは，動物の名前と分類の両方を直接入力する実行プランである．表 2 の規則 4 はこの書き換えを一般化したものである．

[規則 5: 値入力 $\rightsquigarrow$ 判定]

例）与えられた IT 用語の綴りから意味を追加してもらう更新演算を行う実行プランとして，次の実行プランがあるとする．

```
{(ITterm(spell) | ITterm[spell=$spell][?meaning] |
ITterm(spell, meaning))}
```

これは，与えられた IT 用語の綴りから意味を入力する実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(ITterm(spell), Dictionary(spell, meaning) |
ITterm[spell=$spell][?meaning=$meaning] |
ITterm(spell, meaning))}
```

この実行プランは，汎用的な辞書を用いて作成された意味の候補が IT 用語として正しいかどうかを判定する実行プランである．表 2 の規則 5 はこの書き換えを一般化したものである．規則 5 は $X \subseteq Y$ である Y が既知であるときのみ適用可能である．

人による選択演算

人による選択演算は，二種類存在する．一つは，機械で処理可能な属性であるが値が未知で処理不可能なもの．もう一つは，そもそも機械で処理が不可能もしくは困難なもの（与えられた動物が可愛いかどうかの判定，等）．どちらも，条件判定のタスクを行う実行プランと値の入力のタスクを行う実行プランが考えられる．値の入力の場合には，必要に応じて機械で処理できる値，条件とする必要がある．

[規則 6: 未知値入力，条件計算 $\rightsquigarrow$ 判定]

例）高さが未知な山に対して高さが 8000m 以上という選択演算を行う実行プランとして，次の実行プランがあるとする．

```
{(Mountain(name) | mountain[name=$name][?height] |
Mountain(name, height))
(Mountain(name, height), height >= 8000 | null |
HighMountain(name))}
```

これは，まず，与えられた山の高さを入力し，次に，8000m 以上の山を選択するという実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(Mountain(name) | Mountain[name=$name][?height>=8000]
| HighMountain(name))}
```

この実行プランは，8000m 以上の山を直接入力する実行プランである．表 2 の規則 6 はこの書き換えを一般化したものである．

[規則 7: 具体値入力，条件計算 $\rightsquigarrow$ 判定]

例）可愛い動物を求める選択演算を行う実行プランとして，次の実行プランがあるとする．

```
{(Animal(name) | animal[name=$name][?cuteness] |
Animal(name, cuteness)),
(Animal(name, cuteness), cuteness >= 80 |
null | CuteAnimal(name))}
```

これは，まず，与えられた動物の可愛さを機械的に判定できるように 1 から 100 の値で入力し，次に，可愛さが 80 以上の動物を選択するという実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(Animal(name) | animal[name=$name][?is_cute=true] |
CuteAnimal(name))}
```

この実行プランは，与えられた動物が可愛いかどうかを直接判定する実行プランである．表 2 の規則 7 はこの書き換えを一般化したものである．

人による結合演算

人による結合演算は直積と人による選択演算から成るため，人による選択演算に適用可能な規則を同様に適用可能である．

[規則 8: 値入力，順序計算 $\rightsquigarrow$ 順序判定]

例）動物を可愛さで順序付けする実行プランとして，次の実行プランがあるとする．

```
{(Animal(name) | Animal[name=$name][?cuteness] |
Animal(name, cuteness)),
(Animal(name:n1, cuteness:c1),
Animal(name:n2, cuteness:c2), c1>c2 | null |
IsMoreCute(higher:n1, lower:n2))}
```

これは，まず，与えられた動物の可愛さを機械的に判定できるように 1 から 100 の値で入力し，次に，入力された可愛さから順序を計算するという実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(Animal(name:n1), Animal(name:n2) |
animal[name=$n1].?is_more_cute.[name=$n2] |
IsMoreCute(higher:n1, lower:n2))}
```

この実行プランは，与えられた動物のペアにおいて前者が後者よりも可愛いかどうかを判定する実行プランである．表 2 の規則 8 はこの書き換えを一般化したものである．

[規則 9: 値入力，一致計算 $\rightsquigarrow$ 一致判定]

例）同程度に可愛い動物のペアを求める実行プランとして，次の実行プランがあるとする．

```
{(Animal(name) | Animal[name=$name][?cuteness] |
Animal(name, cuteness)),
(Animal(name:a1, cuteness:c1),
Animal(name:a2, cuteness:c2), c1=c2 | null |
IsSameCuteness(a1, a2))}
```

これは，まず，与えられた動物の可愛さを機械的に判定できるように 1 から 100 の値で入力し，次に，入力された可愛さから同じ可愛さの動物を計算するという実行プランである．この実行プランは次の実行プランに書き換えることができる．

```
{(Animal(name:a1), Animal(name:a2) |
animal[name=$a1].?is_same_cuteness.[name=$a2] |
IsSameCuteness(a1, a2))}
```

この実行プランは，与えられた動物のペアが同程度に可愛いかどうかを判定する実行プランである．表 2 の規則 9 はこの書き換えを一般化したものである．

6. 議　　論

本節では，本論文で示した変換規則について議論を行う．

6.1 直接入力によるタスク数の削減

表 2 に示した各変換規則の適用による実行プラン中の総タスク数の増減について表 3 に示す．例えば，5.2 節の規則 1 を導出した例の書き換え前の実行プランでは，山の名前の入力が n 回行われたとすると高さの判定のタスクも n 回行われることになり総タスク数は $2n$ であるが，書き換え後では総タスク数は n 以下で済み，半分以下となる．また，5.2 節の規則 2 を導出し

た例では，書き換え前は学生を m 回入力，学部を n 回入力とすると判定のタスクが mn 回となり，総タスク数は $m+n+mn$ であるが，書き換え後では，総タスク数は m で済む．

これらの書き換えがタスク数を削減できるのは，目的に対して，より直接的な質問を行うタスクへと書き換えているからである．1 節でも示した公用語がスペイン語である国の一覧を目的とする実行プランでは，国名一覧を作成した後，国の公用語を問い合わせるよりも，公用語がスペイン語である国を直接問い合わせたほうがタスク数は大幅に削減できる．本論文で提案する変換規則の利点は，質問をパス式で記述できれば，より直接的な質問に変換できることである．

6.2 タスクあたりの入力データ量

ワーカが入力するデータ量は，タスクを多様な環境に埋め込む際に問題となる．例えば，2bit の入力であれば，四つの選択肢から回答するようなタスクで実現する事ができる．このようにデータ量が少なくなるようにタスクを最適化することで，図 2 に示すように，より単純なインターフェースで実装でき，多様な環境に埋め込める．

表 2 に示した各変換規則の適用による，実行プラン中でワーカが入力するデータ量の一タスクあたりの平均値の増減について表 3 に示す．例えば，5.2 節の規則 6 を導出した例では，書き換え前は可愛さ（1 から 100）を入力するタスクを行うためそのデータ量は 8bit 程度であるが，書き換え後では，1bit となる．また，規則 8 を導出した例についても同様のことがいえる．

規則 5 はデータ量を削減する変換規則である．これができる理由は，規則適用前の実行プランではワーカが入力していた情報を，規則適用後の実行プランでは既存の情報源から正解を含む候補として作成し，ワーカはそれらについて正しいかどうかの判定だけを行うからである．規則 5 は，汎用性の高いデータを用いることで活用できると考えられる．例えば，写真に地点情報をつけるような実行プランでは，地図というデータを用いることで，直接，地名等を入力してもらう実行プランから，写真と分割した地図との結合を行う実行プランに書き換えることができる．

6.3 そ の 他

一般に，タスク数と，タスクあたりの入力データ量はトレードオフの関係がある．しかし，例えば，規則 6, 規則 7 のように，共にタスク数の変化なしにデータ量を削減できる規則が存在する．なぜなら，これらの規則の適用前の実行プランでは，計算機が判定するのに必要な情報をワーカが入力するのに対し，適用後の実行プランでは，計算機の代わりにワーカが直接判定し，その結果のみを入力するからである．

また，同じ入力データであっても質問文が異なることによって，得られる情報量は異なるという点は注目すべき事である．例えば，入力データとして「エベレスト」があった場合に,「山の名前を入力してください」という質問文と「3000m 以上の山の名前を入力してください」という質問文では，得られる情報量は後者の方が大きい．なぜなら，3000m 以上の山の名前を入力するということは，山の名前を入力すると同時に，入力する山が 3000m 以上かどうかという質問に答えていることに等しいからである．このように，一般的な事柄を表す質問文であるほどタスクから得られる情報量は小さいため，具体的な質問をすることでタスク数を減らすことができる．

表 3　変換規則によるタスク数とデータ量の変動

規則	総タスク数	データ量の平均値
規則 1	減少	増加
規則 2	減少	増加
規則 3	減少	増加
規則 4	減少	増加
規則 5	増加	減少
規則 6	変化なし	減少
規則 7	変化なし	減少
規則 8	増加	減少
規則 9	増加	減少

7. まとめと今後の課題

本論文は，マイクロタスクの書き換えによる最適化の提案を行った．また，そのために，質問文をパス式で記述することを提案した．本論文で提案した実行プランの変換規則はリレーショナル代数演算式の変換等の既知のものだけでなく，これまで議論されていなかった規則も含んだものである．

今後の課題としては，これらの変換規則を利用して，与えられた条件を満たす実行プランの探索を行う仕組みの開発が挙げられる．また，完全な変換規則集合の構築など，理論の整備を行うことも今後の課題である．

謝　　辞

本論文の一部は JST さきがけ「情報環境と人」および科研費基盤研究 (#25240012) の支援による．

文　　献

[1] Amazon Mechanical Turk, https://www.mturk.com/.
[2] M. J. Franklin., D. Kossmann, T. Kraska, S. Ramesh, R. Xin. “CrowdDB: answering queries with crowdsourcing”. SIGMOD 2011: 61-72, 2011.
[3] K. Gonnokami, A. Morishima, H. Kitagawa. “Condition-Task-Store: A Declarative Abstraction for Microtask-based Complex Crowdsourcing”. DBCrowd 2013: 20-25, 2013.
[4] K. Heimerl, B. Gawalt, K. Chen, T. S. Parikh, B. Hartmann. “CommunitySourcing: engaging local crowds to perform expert work via physical kiosks.” CHI 2012: 1539-1548, 2012.
[5] A. Marcus, D. Karger, S. Madden, R. Miller, S. Oh. “Counting with the crowd”. PVLDB 6(2): 109-120, 2012.
[6] A. Marcus, E. Wu, D. Karger, S. Madden, R. Miller. “Human-powered sorts and joins”. PVLDB 5(1): 13-24, 2011.
[7] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, J. Widom. “An overview of the deco system: data model and query language; query processing and optimization”. SIGMOD Record 41(4): 22-27, 2013.
[8] Y. Shinagawa, A. Morishima, S. Nakamura, T. Terada. “Human Computation Environment Embedded in Living Spaces for Crowdsourcing Task Processing”. インタラクション 2014, 2014.