Condition Task Store: A Declarative Abstraction for Microtask-based Complex Crowdsourcing

Kenji Gonnokami¹, Atsuyuki Morishima¹, Hiroyuki Kitagawa¹

University of Tsukuba¹
Introduction

As computer network technologies evolved, *crowdsourcing* became popular in many application domains.

**Crowdsourcing Systems**[1]

Software systems that take the crowdsourcing approach

Two Existing Approaches

1. To provide a library of functions that can be called by programs written in imperative programming languages.
2. To allow SQL queries to invoke microtasks.

The former approach provides large expressive power, while the latter allows declarative description with limited expressive power.

Our proposal

We propose alternative declarative approach to implement complex data-centric crowdsourcing with microtasks.

- Declarative
- Large expressive power
Contributions

1. Alternative approach to declarative crowdsourcing
   The CTS abstraction models crowdsourcing systems as a set of CTS rules.
   1. Naturally extends the task templates which adopted by many crowdsourcing platforms to define microtasks
   2. Allows declarative descriptions of crowdsourcing systems
   3. Has large expressive power

2. Novel criterion to measure the expressive power of programming languages for crowdsourcing
   focuses on the class of interactions with humans we can implement with the language
Importance of “class of interactions”

1. Complex crowdsourcing often requires various types of interactions with humans.
   For example, one of such interactions of crowdsourcing is the iterative collaboration, which is not necessary supported by every existing framework.

2. The class of interactions is closely related to the class of games in game theory:
   Because human behavior is affected by the incentives and rules defined by the game structure, the class represents the size of mechanism design space.
Overview

1. Introduction
2. Contributions
3. Related Work
   1. Existing Abstractions
   2. ECA rule
   3. Game Theory
4. CTS Abstraction
5. Expressive Power
6. Prototype
7. Conclusion
Existing Abstractions (1/2)

1. Imperative programming languages
   TurKit\textsuperscript{[2]} provides a library of functions to define and call tasks from general-purpose programming languages

2. MapReduce-like abstraction
   CrowdForge\textsuperscript{[3]} models a crowdsourcing system as a set of tasks to implement partition, map, and reduce functions

3. Control/data flows
   CrowdLang\textsuperscript{[4]} is a language for describing crowdsourcing systems in terms of basic operators, data items, and control flow constructs.


4. Rule-based abstraction

CyLog\cite{5} is a Datalog-like, rule-based language for describing crowdsourcing systems. It requires programmers to be familiar with programming by Horn clauses even for simple crowdsourcing.

5. SQL-like abstraction

CrowdDB\cite{6}, Qurk\cite{7}, and Deco\cite{8} use SQL to describe crowdsourcing systems. They propose novel query processing and optimization schemes.

---

CTS Rule and ECA Rule

Technically, a CTS rule can be implemented by combining two ECA rules[9]

1. Generates micortasks
2. Stores results in the database

The CTS abstraction provides a higher-level, user-friendly abstraction designed for describing crowdsourcing systems, which has a well-defined semantics and proven large expressive power.

Our discussion on the expressive power is related to game theory. Recently the literature on algorithmic game theory has addressed various aspects involving both algorithms and games, such as complexities of computing equilibrium of games\textsuperscript{[10]}.

To our knowledge, our research is the first to discuss classes of games that can be implemented by abstractions for crowdsourcing.

Overview

1. Introduction
2. Contributions
3. Related Work
4. CTS Abstraction
   1. Task Template
   2. Outline of the CTS Abstraction
   3. Examples
5. Expressive Power
6. Prototype
7. Conclusion
The task template is a popular form for defining and registering microtasks into crowdsourcing platforms.
Task Template (2/2)

The task template is a popular form for defining and registering microtasks into crowdsourcing platforms.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<QuestionForm xmlns="http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2005-10-01/QuestionForm.xsd">
  <Question>
    <QuestionIdentifier>1</QuestionIdentifier>
    <QuestionContent>
      <Text>How many movies have you seen this month?</Text>
    </QuestionContent>
    <AnswerSpecification>
      <FreeTextAnswer/>
    </AnswerSpecification>
  </Question>
</QuestionForm>
```

Example of task template for microtasks of MTurk
Overview of the CTS Abstraction

In the CTS abstraction, a crowdsourcing system is described by a set of Condition Task Store (CTS) rules.

We assume that there exists a relational database. CTS rules read and write data to and from the database.
Example 1: Simple Crowdsourcing System

For each book stored in the Book relation, generate three tasks.

<table>
<thead>
<tr>
<th>bid</th>
<th>title</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Catcher in the Rye</td>
<td>J.D. Salinger</td>
</tr>
<tr>
<td>2</td>
<td>Harry Potter</td>
<td>J.K. Rowling</td>
</tr>
</tbody>
</table>

Tags are given by three workers:

1. Youth
2. Novel
3. Wizard
4. Youth
5. Wizard
6. Novel

Workers are paid 10 cents per task, if any other workers entered the same tag.
CTS Rule of Example 1

<table>
<thead>
<tr>
<th>Condition</th>
<th>Book(bid, title, author)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Please tag the book &quot;$title&quot; written by &quot;$author&quot;</td>
</tr>
<tr>
<td>Generator</td>
<td>Entry(desc:&quot;Tag&quot;, var:tag, type:text)</td>
</tr>
<tr>
<td>Count</td>
<td>3</td>
</tr>
<tr>
<td>Payoff</td>
<td>PayIf(count(Tag(bid, tag))&gt;=2, 10)</td>
</tr>
<tr>
<td>Store</td>
<td>Tag(bid, tag)</td>
</tr>
</tbody>
</table>

Please tag the book "The Catcher in the Rye" written by "J.D. Salinger"

Tag [ ] Submit
### CTS Rule of Example 1

<table>
<thead>
<tr>
<th>Condition</th>
<th>Book(bid, title, author)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Please tag the book &quot;$title&quot; written by &quot;$author&quot;</td>
</tr>
<tr>
<td>Generator</td>
<td>Entry(desc:&quot;Tag&quot;, var:tag, type:text)</td>
</tr>
<tr>
<td>Count</td>
<td>3</td>
</tr>
<tr>
<td>Payoff</td>
<td>PayIf(count(Tag(bid, tag))&gt;=2, 10)</td>
</tr>
<tr>
<td>Store</td>
<td>Tag(bid, tag)</td>
</tr>
</tbody>
</table>

#### Book

<table>
<thead>
<tr>
<th>bid</th>
<th>title</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Catcher in the Rye</td>
<td>J.D. Sallinger</td>
</tr>
<tr>
<td>2</td>
<td>Harry Potter</td>
<td>J.K. Rowling</td>
</tr>
</tbody>
</table>
## CTS Rule of Example 1

<table>
<thead>
<tr>
<th>Condition</th>
<th>Book(bid, title, author)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Question: Please tag the book &quot;$title&quot; written by &quot;$author&quot;</td>
</tr>
<tr>
<td>Generator</td>
<td>Entry(desc:&quot;Tag&quot;, var:tag, type:text)</td>
</tr>
<tr>
<td>Count</td>
<td>3</td>
</tr>
<tr>
<td>Payoff</td>
<td>PayIf(count(Tag(bid, tag))&gt;=2, 10)</td>
</tr>
<tr>
<td>Store</td>
<td>Tag(bid, tag)</td>
</tr>
</tbody>
</table>

Please tag the book "The Catcher in the Rye" written by "J.D. Salinger"
CTS Rule of Example 1

Condition | Book(bid, title, author)
---|---

Task | **Question**
---|---
| Please tag the book "$title" written by "$author"
| **Generator**
| Entry(desc:"Tag", var:tag, type:text)
| **Count**
| 3
| **Payoff**
| PayIf(count(Tag(bid, tag))>=2, 10)

Store | Tag(bid, tag)

examples of task-template generators

**Entry**

- Please enter the name of a good restaurant
- Restaurant Name [Submit]

**Decision**

- Are "Amazon.com" and "Amazon River" the same meaning?
  - [YES] [NO]

**Choice**

- Please rate the restaurant "McDonald's" on a 5-point scale.
  - [1] [2] [3] [4] [5] [Submit]

**Comparison**

- Please check the better Japanese food.
  - [Sushi] [Soba] [Submit]
# CTS Rule of Example 1

<table>
<thead>
<tr>
<th>Condition</th>
<th>Book(bid, title, author)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Question</td>
</tr>
<tr>
<td></td>
<td>Please tag the book &quot;$title&quot; written by &quot;$author&quot;</td>
</tr>
<tr>
<td>Task</td>
<td>Generator</td>
</tr>
<tr>
<td></td>
<td>Entry desc:&quot;Tag&quot; var:tag, type:text</td>
</tr>
<tr>
<td>Task</td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Task</td>
<td>Payoff</td>
</tr>
<tr>
<td></td>
<td>PayIf(count(Tag(bid, tag))&gt;=2, 10)</td>
</tr>
<tr>
<td>Store</td>
<td>Tag(bid, tag)</td>
</tr>
</tbody>
</table>

Please tag the book "The Catcher in the Rye" written by "J.D. Salinger"
### CTS Rule of Example 1

<table>
<thead>
<tr>
<th>Condition</th>
<th>Book(bid, title, author)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Please tag the book &quot;$title&quot; written by &quot;$author&quot;</td>
</tr>
<tr>
<td>Generator</td>
<td>Entry(desc:&quot;Tag&quot;, var:tag, type:text)</td>
</tr>
<tr>
<td>Count</td>
<td>3</td>
</tr>
<tr>
<td>Payoff</td>
<td>PayIf(count(Tag(bid, tag))&gt;=2, 10)</td>
</tr>
<tr>
<td><strong>Store</strong></td>
<td>Tag(bid, tag)</td>
</tr>
</tbody>
</table>

The diagram illustrates the tagging task for the book "The Catcher in the Rye" written by "J.D. Salinger" with the following tags: youth, novel, youth.
## CTS Rule of Example 1

<table>
<thead>
<tr>
<th>Condition</th>
<th>Book(bid, title, author)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Please tag the book &quot;$title&quot; written by &quot;$author&quot;</td>
</tr>
<tr>
<td>Generator</td>
<td>Entry(desc:&quot;Tag&quot;, var:tag, type:text)</td>
</tr>
<tr>
<td>Count</td>
<td>3</td>
</tr>
<tr>
<td>Payoff</td>
<td>PayIf(count(Tag(bid, tag))&gt;=2, 10)</td>
</tr>
<tr>
<td><strong>Store</strong></td>
<td>Tag(bid, tag)</td>
</tr>
</tbody>
</table>

### Task:
- **Question:** Please tag the book "$title" written by "$author"
- **Generator:** Entry(desc:"Tag", var:tag, type:text)
- **Count:** 3
- **Payoff:** PayIf(count(Tag(bid, tag))>=2, 10)
- **Store:** Tag(bid, tag)

### Example:
- Tag "The Catcher in the Rye"
- Tag "J.D. Salinger"
- Tag "youth"
- Tag "novel"
- Tag "youth"
### CTS Rule of Example 1

<table>
<thead>
<tr>
<th>Condition</th>
<th>Book(bid, title, author)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Please tag the book &quot;$title&quot; written by &quot;$author&quot;</td>
</tr>
<tr>
<td>Generator</td>
<td>Entry(desc:&quot;Tag&quot;, var:tag, type:text)</td>
</tr>
<tr>
<td>Count</td>
<td>3</td>
</tr>
<tr>
<td>Payoff</td>
<td>PayIf(count(Tag(bid, tag))&gt;=2, 10)</td>
</tr>
<tr>
<td>Store</td>
<td>Tag(bid, tag)</td>
</tr>
</tbody>
</table>

Please tag the book "The Catcher in the Rye" written by "J.D. Salinger"

Tag | bid | tag  |
----|-----|------|
    | 1   | youth|
    | 1   | novel|
    | 1   | youth|
Incentive Structure of Example 1

<table>
<thead>
<tr>
<th>Worker B</th>
<th>novel</th>
<th>good</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>novel</td>
<td>(10, 10)</td>
<td>(0, 0)</td>
<td></td>
</tr>
<tr>
<td>good</td>
<td>(0, 0)</td>
<td>(10, 10)</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rational workers enter tags that others can easily come up with.
Example 2: More Complex Crowdsourcing

Task 1

Please enter the name of a good restaurant

Restaurant Name [input field] [Submit button]

Task 2

Please rate the restaurant "Le Due Spade" on a 5-point scale.

1 2 3 4 5 [Submit button]

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>rname</td>
<td>value</td>
</tr>
<tr>
<td>Le Due Spade</td>
<td>3</td>
</tr>
<tr>
<td>Le Due Spade</td>
<td>5</td>
</tr>
<tr>
<td>Le Due Spade</td>
<td>2</td>
</tr>
</tbody>
</table>

Avg. > 3
## CTS Rules of Example 2

<table>
<thead>
<tr>
<th>Condition</th>
<th>Task</th>
<th>Question</th>
<th>Generator</th>
<th>Count</th>
<th>Payoff</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Task</td>
<td>Please enter the name of a good restaurant.</td>
<td>Entry(desc:&quot;Restaurant Name&quot;, var:rname, type:text)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Count</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payoff</td>
<td>PayIf(avg(Rating(rname, value), value)&gt;3, 10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Store</td>
<td>Restaurant(rname)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Task 1

Please enter the name of a good restaurant

Restaurant Name [ ] Submit
## CTS Rules of Example 2

<table>
<thead>
<tr>
<th>Condition</th>
<th>Restaurant(rname)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Please rate the restaurant &quot;$rname&quot; on a 5-point scale.</td>
</tr>
<tr>
<td>Generator</td>
<td>Choice(var:value, type: int, items: [1, 2, 3, 4, 5])</td>
</tr>
<tr>
<td>Count</td>
<td>3</td>
</tr>
<tr>
<td>Payoff</td>
<td>Pay(10)</td>
</tr>
<tr>
<td><strong>Store</strong></td>
<td>Rating(rname, value)</td>
</tr>
</tbody>
</table>

### Task 2

Please rate the restaurant "Le Due Spade" on a 5-point scale.

Incentive Structure of Example 2

Rational workers for Task 1 enter the names of restaurants that are likely to receive high ratings.

Worker for Task 1
Task 1 「Enter a name of a restaurant」

Workers for Task 2
Task 2 「Rate the restaurant」

Restaurant

Average rating \( \leq 3 \)
(0, 10, 10, 10)

Average rating \( > 3 \)
(10, 10, 10, 10)

Payoff

Rational workers for Task 1 enter the names of restaurants that are likely to receive high ratings.
Overview

1. Introduction
2. Contributions
3. Related Work
4. CTS Abstraction
5. Expressive Power
   1. Two Criterions
   2. Games as a measure of the expressive power
   3. Comparison
6. Prototype
7. Conclusion
Expressive Power of CTS Abstraction

We propose to use the game concept as a measure of the expressive power of programming languages for crowdsourcing.

The class of games that the language can implement affects the way in which the implemented system can exploit the power of the wisdom of crowd.
Definitions Game Classes

[Definition 1]

$g_{1}$ is a class of games that satisfy the following conditions

1. Every input from a human is not affected by the inputs from others

2. The payoffs are computed by a primitive recursive function of the input values.

Games in $g_{1}$ are called simultaneous games in game theory.
Example of $g_1$

Payoffs are defined for each combination of worker inputs. For example, workers receive payoffs when they enter the same tag.

<table>
<thead>
<tr>
<th>Worker A</th>
<th>Worker B</th>
<th>cat</th>
<th>kitten</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>(10, 10)</td>
<td>(0, 0)</td>
<td></td>
</tr>
<tr>
<td>kitten</td>
<td>(0, 0)</td>
<td>(10, 10)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

「Please enter a tag for a given image」

Payoffs are defined for each combination of worker inputs. For example, workers receive payoffs when they enter the same tag.
Definitions of Game Classes

[Definition 2]

\( g_N \) is a class of programs in which

1. \( N(>0) \) is known in advance
2. each game has at most \( N \)-phases of interactions, each of which asks a worker to enter data
3. at each \( i \)-th phase workers are shown some information based on what was entered in the first to the \( i - 1 \)-th phases
4. payoffs are computed by a primitive recursive function of the entered values

Games in \( g_N \) are called fixed length sequential games in game theory
Example of $g_N (N=2)$

「want to divide cakes into two groups whose total prices are equivalent to each other」

Worker A divide cakes

「(A, B, E, F, G), (C, D, H, I)」
Total 51$  Total 49$

Worker B choose one group

「(A, B, E, F, G)」

Expected Payoff

Maximum expected payoff of Worker A is 50

The price of one group
Definitions of the Class of Games

[Definition 3]

\( g^* \) is a class of programs in which each program

1. executes a sequence of interactions with workers, with the sequence being generated by a primitive recursive function

2. shows workers at each interaction the information computed by a function whose parameters are taken from the results of past interactions

3. payoffs are computed by a primitive recursive function of the entered values

A game in \( g^* \) corresponds to a sequential game whose length is not given in advance.
Example of $g^*$

Write a paragraph that explains "Crowdsourcing".

Fixed total payment

Write

Update

Update

...'...

Complete

「NG」

「NG」

「OK」
Theorems

• [Theorem 1]
  \( g_N (N = a) \) is a proper subset of \( g_N (N > a) \).

• [Theorem 2]
  \( g_N \) is a proper subset of \( g_* \).

• [Theorem 3]
  Assume that we allow Turing machines to interact with humans at any step of its execution.
  Let \( M \) be the set of all such machines.
  Then, \( M \) can implement any \( g \in g_* \).
## Comparison of Expressive Powers

<table>
<thead>
<tr>
<th>Abstraction / Framework</th>
<th>Turing complete</th>
<th>Class of games</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTurk alone</td>
<td>No</td>
<td>$\subseteq g_1$</td>
</tr>
<tr>
<td>CrowdForge</td>
<td>No</td>
<td>$\subseteq g_N$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(generally, $N = 3$)</td>
</tr>
<tr>
<td>CrowdDB / Deco / Qurk</td>
<td>No</td>
<td>$\subseteq g_N$</td>
</tr>
<tr>
<td>CTS Abstraction</td>
<td>Yes</td>
<td>$g_*$</td>
</tr>
<tr>
<td>Imperative programming languages with MTurk API</td>
<td>Yes</td>
<td>$g_*$</td>
</tr>
</tbody>
</table>

CTS Abstraction is *declarative* and has *large expressive power*
Overview

1. Introduction
2. Contributions
3. Related Work
4. CTS Abstraction
5. Expressive Power
6. Prototype
   1. Outline
   2. Generators
   3. Demonstration
7. Conclusion / Future Work
Outline of Crapid

A set of CTS rules

<table>
<thead>
<tr>
<th>Condition</th>
<th>Task</th>
<th>Question</th>
<th>Please enter the name of a novel.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generator</td>
<td>Entry(desc:&quot;Novel Title&quot;, var:title, type:text)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Count</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payoff</td>
<td>Pay(3)</td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>Novel(title)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executable code in CyLog\textsuperscript{[5]}

```
schema:
    Novel(
        title text;
    );
Rule:
    Novel(title)/open;
View:
    ...
```

## Task-template Generators

### Entry

**Please enter the name of a good restaurant**

**Restaurant Name**

![Submit button](submit_button.png)

### Choice

**Please rate the restaurant "Le Due Spade" on a 5-point scale.**

1️⃣ 2️⃣ 3️⃣ 4️⃣ 5️⃣

Submit

### Decision

**Are "Amazon.com" and "Amazon River" the same meaning?**

- YES
- NO

### Comparison

**Please check the better Japanese food.**

- Sushi
- Soba

Submit
Demonstration

Rating Restaurants

<table>
<thead>
<tr>
<th>CTS Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Question</strong></td>
</tr>
<tr>
<td>Please enter the name of a good restaurant.</td>
</tr>
<tr>
<td>Please rate the restaurant &quot;$name&quot; on a 5-point scale.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New CTS Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Condition</strong></td>
</tr>
<tr>
<td><strong>Generator</strong></td>
</tr>
<tr>
<td><strong>Count</strong></td>
</tr>
<tr>
<td><strong>Payoff</strong></td>
</tr>
<tr>
<td><strong>Store</strong></td>
</tr>
</tbody>
</table>

Add
Overview

1. Introduction
2. Contributions
3. Related Work
4. CTS Abstraction
5. Expressive Power
6. Prototype
7. Conclusion / Future Work
Conclusion

- We introduced the CTS abstraction, a declarative approach for implementing complex crowdsourcing with microtasks.
- We also introduced a novel criterion to measure the expressive power of programming languages for crowdsourcing by focusing on the class of games we can implement with the languages.
- We showed that CTS abstraction has large expressive power.

Future Work

The development of various rewriting techniques for the CTS abstraction.

We plan to adapt various optimization techniques for crowdsourcing systems into our context.