The Power of Integrated Abstraction for Data-Centric Human/Machine Computations

Atsuyuki Morishima
University of Tsukuba
mori@slis.tsukuba.ac.jp

Norihide Shinagawa
University of Tsukuba
siena@slis.tsukuba.ac.jp

Shoji Mochizuki
University of Tsukuba
mshoji@slis.tsukuba.ac.jp

ABSTRACT

Humans are recognized as important data sources in data-centric applications today. This paper discusses the potential of integrated abstraction of data-centric human/machine computations, where data provided by people plays an important role. We show the potential by using a language named CyLog, our first attempt to develop such abstractions. In CyLog, the closed world assumption is interpreted in a broader world in which people are included as *rational data sources*, that behave rationally in given games. We argue that such abstractions give us opportunities to appropriately deal with computations not closed in machines.

1. INTRODUCTION

Recently, data-centric applications that exploit human/social computation have emerged, such as GWAPs [1] (e.g., the ESP game), Q&A services (e.g., Yahoo Answers), and many other services that require the power of people. Even in some traditional applications, the power of people is essential. For example, data integration and cleaning require not only the processing of large amounts of data by computers but also help from people (e.g.,[10]). As the examples suggest, data from people is essential to attain some of the difficult goals that computers alone cannot attain. People serve as important *data sources* in such applications.

Existing programming languages, including those for Web/DB domains, have been designed only to describe the behavior of computers, and do not offer tools for modeling people as components of computation. Human computation is out of the scope of the languages; the logic of interaction with people needs to be implemented from the scratch using primitive functions (e.g., GUIs) or crowdsourcing APIs (e.g., Amazon Mechanical Turk or AMT). More important, it is difficult to analyze or predict the expected behavior of the entire system, which includes machine and human activities.

This paper discusses the potential of integrated abstraction of data-centric human/machine computations. A good abstraction serves as a powerful tool both in theoretical research and software development; it can be used to describe and analyze problems without implementation details. The description can be used to derive executable codes written in state-of-the-art implementation languages and frameworks, with guaranteed properties. Alternatively, it can be directly executed by engines along with other codes (Sec. 3).

The significance of abstractions of data-centric human/machine computations is increasing for the following reasons. First, as mentioned, it has been found that aggregating the power of machines and humans is a promising approach for achieving some of the computationally difficult goals. Second, in many of data-centric systems today, computation is not necessarily closed in machines. In fact, some of the problems of data-centric systems in practice do not come from bugs of codes but from human factors [7].

To make the discussion concrete, this paper introduces CyLog [11], as an example of such abstractions, and uses it in scenarios of data integration, acquisition, and search. CyLog is a Datalog-like language that introduces open facts and borrows concepts from the game theory to model people as novel *rational data sources*, in order to provide a principled abstraction for describing, analyzing, and executing such programs. An essential difference from the existing languages, is that CyLog deals with human computation as a first class component and allows us to *design and analyze* the behavior of users, while others give no hints on whether users will behave in the expected manner.

2. EXAMPLES AND THE POTENTIAL

A key idea of CyLog is that it does not limit the scope of the closed world assumption to the stored database so that it can naturally incorporate the processes of interactions with people in the language design. CyLog allows some facts to be *open* in the sense that, when the fact is not stored (cannot be derived) in the database, it tries to extend the world by asking people whether a fact holds in the real world.

Open Facts. Fig. 1 shows a fragment of a CyLog program. The fragment, except for the last line, can be interpreted as a Datalog program with the closed world assumption. (Note: CyLog adopts the named perspective [2]. In the program, "x:y" represents a renaming similar to "x `as` y" in SQL.) That is, the ancestors are computed based on the minimum Herbrand model. Therefore, `Ancestor(pam, pat)` holds but `Ancestor(pam, ann)` does not, because the later is not derived from the facts in the database.

In CyLog, open facts are allowed. For example, the last line in Fig. 1 states that if two persons share one parent, there may be some people who know whether the head fact (an instance of `Parent(P, C)` to state that they share another parent) holds. In addition, open facts can have "open" attributes that do not appear in the rule body (e.g., the

```
Parent(pam, bob)          Parent(bob, pat)
Parent(kate, pat)         Parent(kate, ann)
Ancestor(A, D) <- Parent(P:A, C:D)
Ancestor(X:A, D) <- Parent(P:X, C:A), Ancestor(A, D)
Parent(P, C)/open <-Parent(P,C:Y), Parent(P:Z, C:Y), Parent(P:Z, C)
```

Figure 1: Fragment of a CyLog program

```
(a) Ideal:
StuMember(x) <- SIGMOD(x), Student(x)
StuMember(x) <- DBSJ(x), Student(x)

(b) Reality (only a fragment):
StuMember(x) <- SIGMOD(x), DBSJ(y), x!=y, Equiv(x,y), IsReallyStu(x)
```

Figure 2: Data integration example

```
Data:
  MetadataInput(file, keyword)/open <- File(file)
  Metadata(file, g(file):keyword)/game:g(file) <- File(file)
Game:
  g(file)@time(10): duplicate, {MetadataInput}
```

Figure 3: ESP game

```
Data:
  Restaurant(x) <- Restaurants([x|y])
  Restaurants(y) <- Restaurants([x|y])
  Better(a, b)/open <- Restaurant(a), Restaurant(b), a!=b
  Sort(l, result)/game:crowdrank(l) <- Restaurants(l)
Game:
  crowdrank(l)@deadline(2011/6/7): proportional, {Better}
```

Figure 4: Crowd ranking

`keyword` of the first rule in Fig. 3). The open fact is evaluated by a particular person, or a group of people, which is specified by an optional parameter of `/open` or by a built-in predicate. When the key attributes are not open (i.e., bound in the rule body), the semantics of CyLog accepts only the input supplied by the person who came first. As shown in Sec. 3, the system can wait for people to answer with Web forms or crowdsourcing APIs, or can actively use messaging services, in order to have people evaluate open facts.

Seamless Description of Human/Machine Computations. The power of "open" facts comes from the fact that the data acquisition from people can be uniformly described using the same set of language constructs. Let us see a data integration scenario that is based on a real integration experienced by one of the authors. He needed to merge the member-list databases of two academic societies (ACM SIGMOD Japan Chapter and Database Society of Japan). Ideally, the integration is straightforward, because it is the set union of two member databases (Fig. 2 (a)). However, in reality, the task was difficult, confusing, and error-prone, because of the conflicts, inconsistency, and incompleteness of data. We had to deal with many cases that were different mixtures of db queries and human activities. The number of lines in the procedure document (excluding comments) was 153, including 51 lines for manual procedures in a natural language and 39 SQL queries. The manual procedures include sending emails to members to find out whether they were still students and using our knowledge for the entity resolution. In contrast, we can write the entire integration process in CyLog. Fig. 2 (b) shows only a fragment of the actual process, which is constructed by applying expansion rules to the original query in Fig. 2 (a). In that fragment, we need to perform a manual entity resolution using our knowledge and send an email asking if he is still a student, if `Equiv(x,y)` and `IsReallyStu(x)` are open facts.

The seamless description has a set of benefits. First, CyLog allows us to execute the same program in flexible ways with different mixtures of human/machine computations. In the example scenario, if the definition of student members allowed us to know, based on the stored data, whether a member is still a student, the `IsReallyStu` could be computed by machines with other rule definitions or user-defined functions. They are equivalent (except for who evaluates the part). The feature is interesting in various scenarios. For example, some of the recent applications use tools like Twitter as "human sensors;" [16] utilizes tweets to identify the center of the earthquake. CyLog would help us to write programs independent of what the data sources are. Another example that often occurs in many areas is when a huge system (like a reservation system) is down due to the failure of some of its modules, human workers need to substitute for the system until the system becomes operational. CyLog gives us a chance to avoid such all-or-nothing situations. If we could continue to partially execute the program on alive parts of the system, while allowing people to join the execution of the program, the obtained data would be compatible with the program. And it would be easier to restore and merge the data in order to go back to the normal status when the machines became fully operational.

Player A/Player B	Term A	Term B
Term A	(1,1), Term A	(0,0), NULL
Term B	(0,0), NULL	(1,1), Term B

Figure 5: Payoff matrix for `duplicate` game

Order	Date	Player	Rel	Action
1	10:10am	Kate	MetadataInput	tennis
2	10:11am	Ann	MetadataInput	tennis
3	10:12am	Pam	MetadataInput	ball

Figure 6: Path Table

Game	A player is rewarded when she gives:
Majority	the same value as the others.
First	a value given faster than others.
Unique	a unique value.
BestAnswer	a value that received the highest evaluations.

Figure 7: Some predefined games

Second, open predicates allow us the *separation of concerns*. We use the term *data aspect* to denote a quadruple $\langle\langle \mathcal{I}, \mathcal{D}, \mathcal{M}\rangle, \mathcal{R}\rangle$ in which $\mathcal{I}$ is a set of data sources including not only machinery data sources but also people, $\mathcal{D}$ is the data stored in $\mathcal{I}$, $\mathcal{M}$ is the mapping to represent how $\mathcal{D}$ is stored in $\mathcal{I}$, and $\mathcal{R}$ is a set of rules connecting data in $\mathcal{D}$. Open facts allow data aspects to be written in one module and separated from the other code. This makes it easier to understand and maintain the data aspect. In contrast, with the current db languages, data aspects are written by scattered queries combined with the code in other languages. As shown in Sec. 3, a data aspect support system (DASS) is designed to execute the data aspect written in CyLog and programs in general-purpose languages in a combined way.

Finally, the integrated abstraction helps us check if some of the apparently different programs are equivalent, while it is difficult to confirm that different (and mixed) sets of db queries and procedures written in natural/programming languages are equivalent. Therefore, it provides us with a chance of optimizing and transforming data aspects, not individual queries. For example, the data integration scenario used a set of domain-specific expansion rules to safely construct complex data integration procedures.

Reward System. How can we define the semantics of such data aspects? The problem is that human factors are incorporated in the executions. Since people might lie and they need motivation to participate in the computation, it is difficult to predict the execution results. One possible approach is to consider each human as a *rational* data source. By "rational," we mean that people are assumed to provide data in a way consistent with the expected rewards. Therefore, CyLog is required to have a reward system built-in at the language design level. *Games* are abstract concepts that have been well studied in the literature from both theoretical and practical aspects, and the game theory is known to be useful

when discussing not just real "games" but any system that involves incentive structures [3, 9]. In fact, the game theory has been applied to particular classes of problems [15], such as the design of networks, auctions, and GWAPs. CyLog adopts terms and concepts from the game theory to design and implement the appropriate behavior of rational data sources. Then, the semantics of open facts are defined by the *equilibrium* of the game [17]. This is achieved by payoff matrices with outputs (Fig. 5), which we explain below.

We show two example programs. The first one is a simple version of ESP game [1], in which people (players) provide keywords for given image files. If they match, the players are rewarded. Fig. 3 is a CyLog program for the ESP game, where `Data:` defines rules, and `Game:` defines games. The "`/game:g(file)`" specifies that the head is computed after the game identified by `g(file)` is over. The `g(file)` is a *Skolem function* to identify game instances. In the program, a game instance is created for each `file`. In the `Game:` part, `g(file)@time(10)` specifies that each game is invoked for a given `file` and ends in ten seconds.

At the end of the execution of each game instance (identified by `g(file)`), a special table, called a *path table*, is constructed. The table maintains the provenance (corresponding to a *path* in the game theory) to show how the game reached the last state. In other words, a path table records how players have behaved in the game instance. The table is constructed with the schema P(order, date, player, rel, action) (Fig. 6). Each tuple in the table records when and who gave values for the open attributes of the relations specified in { ... } (i.e., `MetadataInput` in Fig. 3).

The payoffs to players and the output values to be consumed by other rules are computed by *game aggregations* of each path table. For example, Fig. 5 shows a part of the payoff marix of `duplicate` game aggregation (Fig. 3) where each cell shows not only the payoffs, but the output value (only two players and two terms are shown in the figure). Given the path table in Fig. 6, payoffs (1 for `Kate` and `Ann`, and 0 for `Pam`) are given to players and the output value is `tennis` given by `Kate` and `Ann`. Then, the value is consumed by the second rule in Fig 3, in which the Skolem function is also used to denote the value. Assuming that people behave rationally, it is expected that the value is computed by the state of equilibrium with the aggregations. We call such a game with output values a *data game*. Applying game aggregations to path tables give us a simple and flexible way to implement both extensive- and normal-form games [17].

The second example is a crowd ranking service (Fig. 4) where a set of restaurants in a city is sorted by subjective preferences of the crowd. The user can execute the program to find must-go restaurants before visiting the city. The `Better(a, b)/open` asks each person if she prefers `a` to `b`. It can be evaluated, for example, through the AMT. The `Sort(l, result)` simply sorts the list `l` by the number of votes given by the game (the code omitted). Here, `proportional` is a payoff function that pays points according to the proportion of the votes for restaurants. Therefore, there is an incentive for the crowd to vote for the restaurants the others prefer too.

As a final note, even in the integration scenario, data games can be incorporated in the program to improve the quality of the results if more than one person is involved.

Diving into Cybernetic Dataspaces. The integrated abstraction allows us to discuss dataspaces involving machines and humans. First, we can discuss the semantics of such dataspaces. As mentioned, let a data aspect d written in CyLog be $\langle\langle \mathcal{I}, \mathcal{D}, \mathcal{M}\rangle, \mathcal{R}\rangle$. Let $T^*_{\mathcal{R},S}(\langle \mathcal{I}, \mathcal{D}, \mathcal{M}\rangle)$ be an operator to compute the (possibly infinite in CyLog) set of *consequences* [2] for the data aspect with *strategies* S, where strategies describe how the people behave in the given games [17]. Since people in $\mathcal{I}$ are involved in the evaluation, there are many strategies and corresponding sets of consequences of a data aspect. When S is a set of *best strategies*, we call $T^*_{\mathcal{R},S}(\langle \mathcal{I}, \mathcal{D}, \mathcal{M}\rangle)$ be a set of rational consequences of d. We can define the semantics of d, denoted by $sem(d)$, as the collection of all the sets of rational consequences of d.

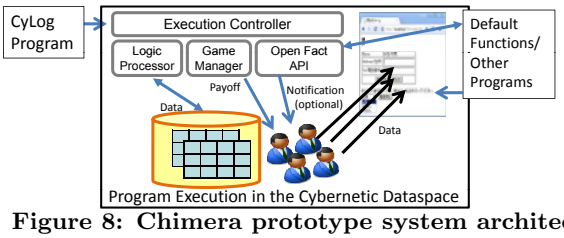

Figure 8: Chimera prototype system architecture

Second, such an abstraction gives us a chance to discuss some properties that we would not discuss when dealing with traditional programs. Although the following theorem seems trivial for CyLog programs, it is clear that such abstractions would promote theoretical developments. A theorem on the efficiency of programs is also discussed in [12].

Theorem 1. *Given a data aspect* $\langle\langle \mathcal{I}, \mathcal{D}, \mathcal{M}\rangle, \mathcal{R}\rangle$ *written in CyLog, there is an algorithm to statically check whether an open fact is not involved in any game definition in* $\mathcal{R}$. □

The theorem is important for the following reason: if an open fact is never involved in any game, it is guaranteed that there is no feedback given to users. Therefore, the program may not be able to continue its execution as intended, or the data given by people may not be appropriate. This is exactly what happened in the Japanese pension system problem [7].

If open facts are involved in games, tools from the game theory can be used to *predict* behavior. For example, the payoff matrix in Fig. 5 is a typical coordination game [17], in which rational players choose the same strategy. Likewise, a simple analysis let us know that the crowd ranking program helps us find popular restaurants, and we can change the game structure to find little known hot spots.

Other Issues. Due to space limitations, we describe other issues briefly. First, CyLog provides programmers with means to implement data games with various settings, e.g., *who* (among the players) and *when* the program should ask whether each open fact holds. Second, some game aggregations are predefined in our library (Fig. 7) but users can provide user-defined game aggregations in the data part. Finally, to explain to people their semantics, predicates can have text descriptions, which can be accessed through APIs provided by the system (as explained next).

3. DATA ASPECT SUPPORT SYSTEMS

DASS should support the execution of the data aspect descriptions (e.g., CyLog programs) by machines *and* people, communicating with programs in general-purpose languages. We discuss one possible set of components and architecture for DASSs by showing those of Chimera, a prototype we developed for CyLog programs (Fig. 8). Chimera adopts a semi-naive evaluation strategy in which the rules are evaluated in a bottom up way [2]. To communicate with other programs, Chimera takes a simple API-based approach; the programs call the open fact API to receive the

data necessary to have people evaluate open facts, and to invoke an event to indicate that a new fact holds.

DASS should support the execution of data aspects *alone*, too, for rapid prototyping. Chimera provides functions to generate Web sites with HTML forms where active evaluation can be realized by messaging services such as emails or Twitter, and plans to provide functions to call the AMT.

4. CHALLENGES AND RELATED WORK

This section identifies some of the new challenges and the related work.

Establishing Theories of Cybernetic Dataspaces. This is definitely one of the most exciting challenges, which can lead to significant contributions. An important issue is how to design dataspaces involving human activities, with guaranteed properties. We showed only a first step, and there are many interesting questions. For example, given a program and different mixtures of human/machinery computers ($\langle\langle\mathcal{I},\mathcal{D},\mathcal{M}\rangle,\mathcal{R}\rangle$ and $\langle\langle\mathcal{I}',\mathcal{D},\mathcal{M}'\rangle,\mathcal{R}\rangle$), how different the efficient optimizations are? Given two different programs involving human activities to achieve the same goal (two data aspects $d_1 : \langle\langle\mathcal{I},\mathcal{D},\mathcal{M}\rangle,\mathcal{R}\rangle$ and $d_2 : \langle\langle\mathcal{I},\mathcal{D},\mathcal{M}\rangle,\mathcal{R}'\rangle$ s.t. $sem(d_1) = sem(d_2)$), which one reaches the goal faster? How much payoffs are needed to reach the goal of a program in general? Are there design criteria or normal forms for cybernetic dataspaces? We believe that data-centric abstraction is promising to establish theories.

Of course, games are not a magic wand; in some applications, it may be difficult to provide real benefits (e.g., points, money, and evaluation scores) to be modeled by payoff values. Humans are not necessarily rational. However, we believe that modeling humans as rational data sources to apply the game theory is a good starting point. An interesting open question is whether we can apply the results from other fields such as cognitive and behavioral sciences.

Assignment of Computations to Appropriate Machines and Humans. Humans are not homogeneous in their abilities. In CyLog, it is programmers that give decide who provides the required data (by specifying $\langle\mathcal{I},\mathcal{D},\mathcal{M}\rangle$), but finding appropriate humans to evaluate open facts is an important open problem.

Data Aspect Issues. Since open facts allow us to describe not only individual queries but also data-oriented human activities as a data aspect, we now have an abstraction for wider data-centric issues. Although an event-based interface is fine to connect the data aspect with other programs, it is still an open problem to identify the best interface.

Language Design. CyLog adopts Datalog as a basis for the following reasons. First, as recent studies suggest [8], languages based on Datalog allow us to concisely describe data-centric applications. Second, the rule syntax has good compatibility with user inputs (e.g., [4]). Finally, logic-based programming has an affinity toward event-driven executions, which data games often require. Although open predicates and data games are key components of CyLog since the beginning [11], other important constructs and the good overall design of the language are still open problems.

Related Work. Qurk [13], hQuery [14] and CrowdDB [6] are independently conducted research projects but closely related to ours. Their focus is to achieve database functions on the crowd, and one of the interesting points is to attain *data independence* in the presence of human data sources, on the assumption that a crowd of people are supplied by crowdsourcing services. Currently, they take the *crowd-as-a-data-source* approach, in the sense that their focus is on how to provide programmers with the view of a reliable data source over the underlying set of people as a whole.

We believe that CyLog is unique in that we take the *human-as-a-data-source* approach, in which each human (player) is modeled as a visible data source, and we try to design data-centric abstractions to deal with the "new" type of data source for building cybernetic dataspaces in flexible ways. We introduce the concept of rational data sources and give new components, such as data games, to interact with rational data sources. We believe that data-centric human/machine computations are important in many scenarios [5] beyond the AMT-style crowdsourcing setting which provides only limited forms of game situations. Dealing with dataspaces with other styles of human involvements are not out of the scope of CyLog. It is interesting, however, that those projects including CyLog employ many overlapped concepts. For example, hQuery is discussed with a Datalog style notation. CNULL [6] and our open attribute values are similar to each other. The "majority votes" principle, discussed in most of the crowd-as-a-data-source researches, is related to coordination games in CyLog. Some of the challenges discussed here are also addressed in the projects.

Acknowledgement. The authors are grateful to Prof. Sugimoto, Prof. Sakaguchi, Prof. Nagamori, and Prof. Wuwongse for the discussion in seminars, and Prof. Tajima and Prof. Kitagawa for giving us valuable comments on earlier versions of the paper. This research is supported by PRESTO from the Japan Science and Technology Agency.

5. REFERENCES

[1] L. von Ahn, L. Dabbish: Designing games with a purpose. CACM 51(8): 58-67 (2008)
[2] S. Abiteboul, R. Hull, V. Vianu. Foundations of Databases, Addition-Wesley, 1995.
[3] G. Anthes: Mechanism design meets computer science. CACM 53(8): 11-13 (2010)
[4] X. Chai, B. Vuong, A. Doan, J. F. Naughton: Efficiently incorporating user feedback into information extraction and integration programs. SIGMOD Conference 2009: 87-100
[5] A. Doan, R. Ramakrishnan, A. Y. Halevy: Crowdsourcing systems on the World-Wide Web. CACM 54(4): 86-96 (2011)
[6] M J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, R. Xin: CrowdDB: answering queries with crowdsourcing. SIGMOD 2011: 61-72
[7] Ministry of Internal Affairs and Communications, Japan. The report on the pention system problem. http://www.soumu.go .jp/menu_news/s-news/2007/071031_3.html.
[8] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic SIGMOD Record , 39(1), 2010.
[9] S. Jain, D. C. Parkes: The role of game theory in human computation systems. KDD Workshop on Human Computation 2009: 58-61
[10] M. Keulen, A. Keijzer: Qualitative effects of knowledge rules and user feedback in probabilistic data integration. VLDB J. 18(5): 1191-1217 (2009)
[11] A. Morishima: A Database Abstraction for Data-intensive Social Applications. The 5th Korea-Japan Database Workshop 2010 (KJDB2010), May 28-29, 2010. (slides available)
[12] A. Morishima, N. Shinagawa: The Power of Integrated Abstraction for Data-Centric Human/Machine Computations. Technical Report, 2011.
[13] A. Marcus, E. Wu, S. Madden, R. C. Miller: Crowdsourced Databases: Query Processing with People. CIDR 2011: 211-214
[14] A. G. Parameswaran, N. Polyzotis: Answering Queries using Humans, Algorithms and Databases. CIDR 2011: 160-166
[15] Y. Shoham: Computer science and game theory. Commun. ACM 51(8): 74-79 (2008)
[16] T. Sakaki, M. Okazaki, and Y Matsuo: Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors, WWW2010.
[17] F. Vega-Redondo. Economics and Theory of Games, Cambridge University Press, 2003.