UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1

Ecole Doctorale N°601
Mathèmatique et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*
Par

« Rituraj SINGH »

« Data Centric Workflows for Crowdsourcing Application »

Thèse présentée et soutenue à RENNES, le 07-05-2021
Unité de recherche : INRIA, IRISA (UMR 6074), Rennes – Bretagne Atlantique

Rapporteurs avant soutenance :

Salima BENBERNOU	Professeur, HdR	Université Paris Descartes
Farouk TOUMANI	Professeur, HdR	Blaise Pascale University

Composition du jury :

Président :	Stefan HAAR	Directeur de recherche	INRIA Paris – Saclay
Examinateurs :	Albert BENVENISTE	Directeur de recherche	INRIA Rennes – Bretagne Atlantique
	Marco MONTALI	Associate professor	Free University of BozenBolzano
Dir. de thèse :	Loïc HÉLOUËT	Chargé de recherche	INRIA Rennes – Bretagne Atlantique
Co-encadrant :	Zoltán MIKLÓS	Maître de conférences	l'Université de Rennes

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisors Loïc Hélouët and Zoltán Miklós for being the ideal supervisor a doctoral student can hope for. Loïc is absolutely brilliant with his astute logical thinking with precise insights and Zoltán's ability to give crisp feedback with his razor-sharp intellect is unmatched. They allowed me absolute freedom and encouraged me to explore new ideas, provided tools and methodologies to perform research and actively sought out to listen to my half baked rumbling ideas. I remember my first day when Loïc told me whenever I want, I can come and knock on his office and shamelessly I knocked on his office so many times that I even cannot keep track of it. It was very kind of Loïc to keep the doors always open and available. Thank you, Zoltán for bearing me in your office for three years- from asking lame questions anytime to whiteboard discussions. I will miss sharing the office with you. Apart from being academic sound, they are very nice human being. From my initial days in France, they helped in every possible way, ranging from my residence permit to university forms, reviewing and teaching, all the way to taking care of defence. I am indebted for the kind advice forever that I received in the last three and half years and I'm sure I have yet to learn so much more from them.

Next, my thanks go to the members of the defense jury for the thesis who did me the honour of agreeing to review it: Salima Benbernou and Farouk Toumani for their reports and relevant feedback, Stefan Haar, Albert Benveniste and Marco Montali for kindly evaluating this work along with their support during the last few months. I would also like to thank my CSID members Emmanuelle Anceaume and Matthias Weidlich for yearly progress review and evaluations.

I would like to express my sincere gratitude to David Gross-Amblard for his kind support and help. David is quite instrumental in the implementation and coordination of the HEADWORK project. Apart from that from the beginning, David helped me a lot to integrate me with the DRUID team - Thank you! I will always remember and cherish the time spent with David - from project to development meetings, from lunch talks to movies. I would also like to express my sincere thanks to Pierre Bourhis for his help in proofs and collaboration.

I was fortunate enough that during my PhD days, I was part of the DRUID and SUMO team, with such a nice bunch of friendly researchers. Both team members are very friendly and cooperative. I will forever cosset the moments from team lunch to the green seminar. To single out, I will miss talking with Tristan Allard, Mickaël Foursov, Arnaud Martin, Nathalie Bertrand, Ocan Sankur, Hervé Marchand, Thierry Jéron, Abd-El-Karim Kecir, Adrian Aubel, Ian Jeantet, Joris Duguépéroux and Tompoarinaina Andriamilanto (Napoun). Especially, I will miss coffee breaks with Ian and our long talks on several contemporary topics.

This academic pursuit will be incomplete without taking the names of friends who were always there in my ups and downs. I would like to thank my buddies- Arif, Jayabrata, Suman, Neetu, Anirban, Arun, Soumen, Arpan, Ayan, Subhodeep, Subarna, Shabina and Pooja that I met in Rennes and in a short time they became an indispensable part of my life. Thank you for making it happening in Rennes. I would like to thank my close-knit group of IIT Patna friends- Subho, Debapriya, Ritesh, Nilesh, and Sudipta. I would also like to thank my amazing friend- Pratyusha, Chetanya, Ved Prakash, Vikas - many thanks, guys! I would also like to thank my bunch of school friends- Himanshu, Anirudh, Jainath, Nitesh, Parimal and Ved who always made me feel at home even when I was thousands of miles away. You all are amazing guys and much love to you all.

Finally and most importantly, my heartfelt gratitude to my parents- Ranjit and Rani who is the sole source of energy behind everything in my life. My father, himself an academician is my inspiration for pursuing my doctoral studies. My mother unbounded love and sacrifice cannot be described in words. I would also like to thank my brother Keertiraj and cousins Abhishek, Anamika and Poornima for their boundless support and care.

I would like to close this section by thanking all the institutions and agencies that made the realization of this work, namely: Agence nationale de la recherche (ANR), HEADWORK project (ANR-16-CE23-0015) [1], Wirk [2], Muséum national d'Histoire naturelle-participatory science project SPIPOLL [3], the University of Rennes I (UR1), Institut national de recherche en sciences et technologies du numérique (INRIA) and Institut de recherche en informatique et systèmes aléatoires (IRISA).

1. http://headwork.gforge.inria.fr/
2. https://www.wirk.io/
3. https://www.spipoll.org/

RÉSUMÉ

Mots-clés: Crowdsourcing, workflow centrés sur les données, assurance de la qualité

Introduction

Avec la génération exponentielle de nouvelles données, les organisations sont confrontées à une surcharge d'informations pour gérer les données, concevoir des algorithmes et extraire des connaissances. La plupart des données produites sont non structurées et se composent d'images, de vidéos et de textes en langage naturel. Elles doivent être intégrées, traitées, stockées, indexées, exploitées, suivies et rapportées pour répondre aux besoins d'une organisation sur des marchés concurrentiels conscients des données. Bien que l'organisation s'appuie sur des mécanismes automatisés alimentés par des algorithmes de gestion de données et d'intelligence artificielle, l'automatisation complète est encore loin d'être acquise. En outre, certaines tâches sont mieux exécutées par des humains, d'autres sont centrées sur l'homme (par exemple, les tâches de sondage, la collecte de données, etc.). Dans ces cas, l'homme peut jouer un rôle clé dans l'analyse des données, ce qui nécessite une intelligence naturelle.

Les humains sont capables de comprendre les données sous toutes leurs formes - texte, image et vidéo. L'accès généralisé à l'internet a ouvert la voie à l'utilisation de la sagesse de la foule pour traiter les données. Le crowdsourcing est apparu comme un nouveau paradigme majeur pour résoudre les tâches qui requièrent l'intelligence humaine et qui sont difficiles à résoudre par des machines. Par exemple, les tâches de résolution d'entités, de reconnaissance d'images, d'analyse des sentiments peuvent être mieux exécutées en utilisant des acteurs humains. Des plateformes commerciales comme Amazon Mechanical Turk (AMT), Crowdflower, Foule Factory, etc. offrent un moyen facile d'engager des internautes et de les récompenser.

En général, toutes les plates-formes de crowdsourcing suivent le même workflow d'exécution. Un client a une tâche et est prêt à l'exécuter sur une plate-forme de crowd-

sourcing. Le client soumet la tâche à la plate-forme. La plate-forme de crowdsourcing attribue ensuite la tâche à plusieurs travailleurs. Les travailleurs exécutent la tâche en utilisant leur expertise et leurs convictions, puis renvoient les résultats à la plate-forme. La plate-forme regroupe, traite les résultats et, en retour, les travailleurs reçoivent unè rècompense. Après le traitement, les résultats sont renvoyés au client par la plate-forme. Le crowdsourcing est utilisé dans divers domaines comme l'intelligence artificielle pour recueillir des données d'entrainement, le traitement de texte, les enquêtes, le marketing en ligne, etc.

Problèmes et objectif de la thèse

La plupart des plateformes de crowdsourcing actuelles telles que l'AMT permettent la réalisation d'un large éventail de tâches. Les tâches considérées sont principalement des micro-tâches. Les micro-tâches sont des tâches de petite taille, rapides et indépendantes qui nécessitent peu de temps de réalisation (tagger une image, répondre à une simple question booléenne, etc.). Cependant, dans la vie réelle, les tâches sont souvent complexes et nécessitent plusieurs niveaux d'expertise. Considérons une tâche complexe: Acquérir des images d'insectes, juger de la qualité de l'image, annoter l'image de haute qualité en utilisant une taxonomie prédéfinie, puis rédiger une brève description pour chaque image. Ces tâches complexes sont un véritable défi. Tout d'abord, un modèle permettant de définir une tâche complexe en termes d'orchestration de tâches plus petites est nécessaire. Il demande de spécifier la tâche ainsi que d'orchestrer sa réalisation. L'orchestration n'est pas seulement une question d'ordonnancement des micro-tâches : Les données doivent être transmises d'une micro-tâche aux suivantes. De plus, l'ensemble du processus d'exécution doit se terminer par un ensemble correct de résultats. Les réponses fournies par les foules sont subjectives et sujettes à erreur. Pour atténuer le problème et maximiser l'exactitude des résultats, les tâches sont généralement réalisées par plusieurs personnes. C'est pourquoi les réponses provenant de différentes sources doivent être agrégées. En outre, une tâche complexe est assortie d'un budget prédéfini qui permet d'engager des travailleurs et de les récompenser pour la réalisation de la tâche. Certaines tâches peuvent nécessiter quelques réponses pour parvenir à un accord, tandis que d'autres tâches complexes (par exemple, les tâches demandant des opinions ou utilisant les croyances du travailleur) peuvent nécessiter plus de réponses pour construire un résul-

tat précis. Il faut un mécanisme permettant de dépenser le budget de manière optimale et de trouver un compromis entre le coût et la précision. Par conséquent, la complexité du déploiement de la tâche complexe ainsi que l'optimisation du coût et de la précision sont en corrélation de manière complexe, ce qui rend difficile l'optimisation des compromis entre eux lors du traitement des données et de la conception d'un algorithme.

L'objectif de la thèse est de définir des techniques permettant de déployer des applications complexes en plus des plateformes de crowdsourcing conventionnelles et de fournir des algorithmes centrés sur les données optimisant le coût et la précision. Nous relevons ces défis en définissant d'abord des modèles formels pour des workflow complexes et en fournissant ensuite des modèles probabilistes pour gérer le compromis entre le coût et la précision. Nous fournissons également un outil permettant de vérifier les propriétés de terminaison et d'exactitude des workflow complexes.

Modèle pour les workflow complexes

Outre les simples tâches de renseignement humain telles que l'étiquetage des images, l'analyse des sentiments, les plateformes de crowdsourcing ont la capacité de réaliser des tâches plus complexes. L'étape suivante du crowdsourcing consiste à concevoir des processus complexes en s'appuyant sur les crowds existants. En effet, de nombreux projets, et en particulier les workflows scientifiques, prennent la forme d'orchestrations de tâches composites de haut niveau. Chaque tâche de haut niveau peut être considérée individuellement comme une tâche de collecte de données, ou comme le traitement d'un grand ensemble de données, construit comme l'union des résultats de micro-tâches faciles indépendantes. Toutefois, la coordination de ces tâches de haut niveau pour atteindre l'objectif final nécessite des processus plus évolués. On peut facilement rencontrer des situations dans lesquelles le résultat d'une tâche de haut niveau sert d'entrée pour l'étape suivante du processus global : par exemple, on peut vouloir retirer d'un ensemble de données des images de mauvaise qualité avant de demander aux utilisateurs de les annoter. De même, certaines situations permettent un traitement parallèle de l'ensemble de données suivi d'une fusion des résultats obtenus. Un exemple typique est la validation croisée des réponses renvoyées par différents travailleurs du crowd.

De nombreux projets ne peuvent être décrits comme des collections de microtâches répétitives et indépendantes : ils nécessitent des compétences spécifiques et une col-

laboration entre les participants. Nous appelons de tels projets “complex tasks”. La forme typique des tâches complexes est une orchestration de phases de haut niveau. Chacune de ces phases requiert des compétences spécifiques, peut être considérée à son niveau comme un nouvel objectif en soi et peut être décomposée en chorégraphies plus fines, jusqu'au niveau de l'assemblage de micro-tâches. Le déroulement de ces processus est donc dynamique et doit tenir compte des compétences des travailleurs, de leur disponibilité et des données de sortie produites, mais aussi de leurs connaissances sur les processus eux-mêmes. Le premier défi consiste à combler l'écart entre un processus de haut niveau qu'un demandeur souhaite réaliser et sa mise en œuvre en termes de composition de micro-tâches. Passer d'un niveau de description à l'autre n'est pas facile, et nous préconisons l'utilisation de l'expertise de la foule pour un tel raffinement. Cela peut être réalisé avec des réponses d'ordre supérieur, permettant à un travailleur bien informé de renvoyer une orchestration de tâches plus simples au lieu d'une réponse nette à immédiate question.

La première contribution de la thèse est un modèle de workflow centré sur les données, appelé workflow complexe, pour spécifier, vérifier et déployer des tâches complexes sur une plate-forme de crowdsourcing existante. Le modèle fournit des constructions de haut niveau qui permettent la conception de tâches complexes, décrites comme une orchestration d'un ensemble de tâches simples, et gère en outre les compétences des travailleurs, la dépendance aux données et les contraintes liées aux tâches. Il permet l'exécution de tâches, qui sont soit des tâches automatisées qui transforment des ensembles de données, soit des tâches réalisées par un travailleur. En outre, les travailleurs peuvent proposer de raffiner une tâche complexe qui semble trop complexe pour être réalisée par un seul travailleur de la foule. Ce raffinement est spécifié comme des actions d'ordre supérieur qui permettent de remplacer une tâche par un workflow fini. Nous avons défini la syntaxe et la sémantique du modèle. Les tâches sont classées en trois catégories : les tâches atomiques (peuvent être réalisées en une seule étape par le travailleur), les tâches complexes (doivent être décomposées en une orchestration de tâches plus petites) et les tâches automatisées (tâches réalisables par la machine). Un workflow complexe est étiqueté comme un graphe acyclique dirigé où chaque nœud est mis en correspondance avec une tâche et les bords représentent la relation de priorité sur l'exécution des tâches. Nous fournissons ensuite quatre règles sémantiques qui servent de principe directeur pour le workflow. Les règles sémantiques définissent : l'attribution de tâches à des travailleurs libres, l'exécution d'une

tâche atomique par un travailleur, l'exécution d'une tâche automatisée, et formalise le raffinement. Le raffinement d'un nœud marqué à une tâche complexe par un travailleur remplace le nœud par un nouveau workflow qui contient un ensemble de tâches remplissant de manière composite l'objectif du nœud raffiné. L'exécution d'un workflow complexe consiste en l'application de tâches selon l'ordre prescrit par le workflow. Les tâches prennent en entrée les ensembles de données produits par leur prédécesseur dans le workflow, et produisent de nouveaux ensembles de données, ou affinent le workflow actuel.

Décidabilité

Un workflow complexe est défini par un ensemble de règles sémantiques pour répartir les travailleurs, orchestrer et exécuter les tâches. Cependant, un workflow peut ne jamais atteindre une configuration finale. Cela peut être dû à la saisie de données particulières par les travailleurs qui ne peuvent pas être traitées correctement par le workflow, ou à une réécriture infinie apparaissant pendant l'exécution. Dans de tels cas, le workflow peut se trouver dans une impasse. Cette impasse bloque le flux d'exécution du workflow et empêche d'atteindre l'objectif final. Même lorsqu'un workflow termine toujours, cette propriété seule ne suffit pas à répondre à l'exigence du client. Par exemple, un workflow W peut se terminer, mais avec un mauvais ensemble de résultats, c'est-à-dire des résultats qui ne sont pas conformes aux exigences du client. Dans ce cas, la sortie retournée n'est d'aucune utilité pour le client. Il est donc important de garantir l'exactitude du workflow en même temps que sa terminaison.

Nous examinons les propriétés formelles du modèle, en commençant par la question de la terminaison: Étant donné un workflow complexe, un ensemble de travailleurs avec leurs profils et les transformations de données sous-jacentes, un workflow se termine-t-il pour au moins une seule exécution (terminaison existentielle) et pour toutes les exécutions d'un workflow (terminaison universelle) ? Nous avons établi que la terminaison existentielle est en général indécidable en raison de la partie contrôle du workflow (les workflow complexes peuvent simuler deux contre-machines). D'autre part, la terminaison universelle est décidable, et nous avons présenté un sous-ensemble intéressant du modèle pour lequel la terminaison existentielle est décidable. Plus précisément, nous limitons le nombre de raffinements des tâches qui peuvent se produire pendant l'exécution d'un workflow complexe et supposons que le workflow n'a pas

de réécriture récursive des tâches. Ensuite, la terminaison se résume à la réalisation de dépendances de données dans un ensemble fini d'exécutions du workflow. Nous donnons un algorithme pour vérifier la terminaison d'un workflow complexe avec une récursion limitée basée sur une dérivation de plus faible précondition. Les plus faibles préconditions ont été introduites dans [Dij75] et constituent un moyen formel de prouver l'exactitude d'un programme. Nous montrons que des fragments de FO sont fermés sous le calcul de précondition. Ensuite, nous examinons la question de la correction d'un workflow, qui est satisfaite si un worklow se termine et que le résultat produit répond aux contraintes des données fournies par le client. Encore une fois, si les contraintes de sortie sont exprimées en fragments de FO décidables, correctin d'un workflow complexe est décidable. Nous présentons également l'analyse de la complexité de la terminaison et de la correction du workflow. Nous constatons que la complexité en (co)-2EXPTIME pour les fragments ayant la complexité la plus faible provient principalement de la taille exponentielle de la formule décrivant les préconditions qui doivent être satisfaites dans la configuration initiale. Cela peut être considéré comme une complexité intraitable, mais on peut cependant s'attendre à ce que la récursion dans les workflows complexes soit bornée (avec une borne assez petite) ce qui devrait rendre les analyses de terminaison et correction faisables.

Assurance Qualité pour les Tâches Atomiques

Le modèle de workflow complexe proposé permet de spécifier des tâches complexes avec des workflow et peut vérifier l'exactitude, la terminaison sur un sous-ensemble raisonnable du modèle, principalement des spécifications non récursives. Un workflow fournit un moyen efficace de synchroniser des tâches complexes sous la forme de différentes phases pour atteindre les objectifs d'un client. Ils définissent la manière dont les tâches sont décomposées, ordonnées et exécutées. Cependant, ils ne fournissent pas de mécanismes pour garantir la qualité des données produites par le workflow. Le processus de vérification ne permet pas non plus de prendre en compte le coût d'un workflow. En général, les tâches de la plate-forme de crowdsourcing sont assorties d'un budget fixe fourni par le client. Nous considérons d'abord le coût et la qualité des données produites pour une seul tache atomique consistant à analyser des données dans un dataset, et où chaque enregistrement peut être étudié par plusieurs contributeurs.

Dans le crowdsourcing, les travailleurs sont très hétérogènes : ils ont des origines différentes, un domaine d'expertise. Comme nous ne pouvons pas faire confiance à un seul contributeur et pour faire face à cette hétérogénéité, les tâches sont souvent reproduites. Un problème fondamental consiste à déduire une réponse correcte à partir de l'ensemble des résultats renvoyés. Un autre défi est d'obtenir une réponse fiable à un coût raisonnable. Un budget illimité permet de disposer d'un grand nombre de travailleurs pour chaque tâche, mais un budget limité oblige à utiliser au mieux les ressources.

Nous proposons une technique d'agrégation pour les plateformes de crowdsourcing. Nous considérons les deux facteurs clés que sont la difficulté de la tâche et l'expertise des travailleurs, exprimés en termes de rappel et de spécificité pour modéliser la réponse à une tâche produite par un travailleur. L'agrégation est basée sur l'algorithme d'Expectation Maximization [DLR77] et estime conjointement les réponses, la difficulté des tâches et l'expertise des travailleurs. Parallèlement, nous proposons également CrowdInc, une technique d'étiquetage itérative qui optimise le coût global de la collecte des réponses et de leur agrégation. L'algorithme met en œuvre une politique de répartition des travailleurs qui prend des décisions à partir d'un seuil de qualité dynamique calculé à chaque cycle, ce qui permet d'obtenir un bon compromis entre le coût et la précision. Nous évaluons l'algorithme sur des ensembles de données réels pour valider notre approche. Nous montrons sur plusieurs points de référence que CrowdInc atteint une bonne précision, réduit les coûts, et nous comparons ses performances aux solutions existantes.

Assurance Qualité pour les Workflow Complexes

Après avoir défini un algorithmeun algorithme permettant d'optimiser le coût et la précision pour une seule phase, nous étendons l'algorithme de compromis entre le coût et la précision conçu au chapitre précédent aux workflows complexes. Une tâche dans un workflow peut être réalisée de deux manières, à savoir l'exécution synchrone (une tâche traite l'intégralité de ses données d'entrée avant d'envoyer le résultat à la suivante) et asynchrone (une tâche envoie à la suivante des données dès qu'elles sont prêtes). Nous ajoutons la notion de phase à notre modèle et nous en revoyons la sémantique pour tenir compte de la réplication des tâches et de l'agrégation des réponses. Nous étudions ensuite les défis posés par ces deux types d'exécution. Nous

utilisons l'algorithme d'agrégation pour une tâche atomique dans le cadre d'un workflow. Nous montrons que différents mécanismes de seuil sont nécessaires pour les deux types d'exécution. L'exécution synchrone nécessite une politique de seuil locale qui dérive un seuil par rapport aux tâches et au budget alloué à une phase. En revanche, l'exécution asynchrone nécessite une politique de seuil globale qui détermine le seuil par rapport à toutes les tâches des différentes phases et au budget total. Enfin, nous comparons nos résultats avec les approches classiques de crowdsourcing et nous constatons que la technique proposée permet d'obtenir au moins la même précision qu'une technique utilisant la majorité en mode synchrone avec un budget réduit. Dans la plupart des contextes, cependant, l'approche proposée permet d'économiser une partie du budget. Nous présentons l'analyse des coûts et de la précision et les résultats dans différentes configurations.

Platform

La dernière contribution de cette thèse est un outil appelé Crowdplex qui implémente les algorithmes d'analyse de terminaison et de correction présentés dans les chapitres précédents

TABLE OF CONTENTS

Acknowledgements i

Résumé iii

I Prologue 1

1 Introduction 2
1.1 Thesis Overview and Contribution 5
1.2 Outline 9

2 State of the Art 11
2.1 Crowdsourcing Marketplaces 11
2.2 Crowdsourcing Space 14
2.3 Orchestration of Tasks and Languages 17
2.3.1 Process Centric Approaches 18
2.3.2 Artifact Centric Approaches 27
2.4 Data 28
2.4.1 Reasoning on Data 30
2.4.2 Datalog 35
2.4.3 SQL 37
2.4.4 Data Centric Models 40
2.4.5 Weakest Precondition 42
2.5 Quality Assurance 44
2.5.1 Aggregation Techniques 45
2.5.2 Budget optimization 50

II Data Centric Workflows for Crowdsourcing 55

3 Complex Workflows for Crowdsourcing 56

3.1 Higher Order Example . . . 57
3.1.1 A simple example: the actor popularity poll . . . 58
3.1.2 A real field example: the SPIPOLL initiative . . . 59
3.2 Preliminaries . . . 61
3.3 Workflow Formalization . . . 64
3.4 Operational Semantics . . . 67
3.4.1 Data operations . . . 67
3.4.2 Operational Semantics . . . 70
3.5 Conclusion . . . 77

4 Decidability **79**
4.1 Effective Computation of Weakest Preconditions . . . 80
4.1.1 Closure of FO classes . . . 82
4.2 Termination of Complex Workflows . . . 86
4.2.1 Symbolic Execution Tree . . . 96
4.2.2 Termination with a guaranteed bound . . . 97
4.3 Correctness of Complex Workflows . . . 101
4.4 Use Case . . . 107
4.4.1 Formulation of Symbolic Execution Tree . . . 108
4.4.2 Algorithm to check termination . . . 110
4.5 Platform . . . 113
4.6 Conclusion . . . 114

III Quality Assurance **117**

5 Quality Assurance for Atomic Tasks **118**
5.1 Basic ingredients of aggregation . . . 121
5.1.1 Probability theory . . . 121
5.1.2 Factors influencing efficiency of crowdsourcing . . . 122
5.1.3 Expectation Maximization . . . 124
5.2 The Aggregation model . . . 125
5.2.1 Aggregating Answers . . . 128
5.3 Cost Model . . . 130
5.3.1 Confidence and Threshold . . . 131

5.3.2 CrowdInc: An algorithm to optimize costs 132
5.4 Experiments 135
5.5 Conclusion 139

6 Quality Assurance for Complex Workflows 141
6.1 Introduction 141
6.2 Complex workflow with aggregation 143
6.3 Aggregation Model 153
6.4 Cost Model for Workflow 153
6.5 Experiments and results 163
6.6 Conclusion 171

IV Closure 173

7 Conclusion 174
7.1 Contribution Summary 174
7.2 Perspectives 175
7.2.1 Short Term Perspectives 176
7.2.2 Long Term Perspectives 178

Bibliography 183

Appendix 202
A.1 Proof of Theorem 1 202
A.2 Proof of Theorem 2 206

LIST OF FIGURES

1.1 Crowdsourcing mechanism. 3

2.1 Screenshot from Amazon Mechanical Turk. 12
2.2 Screenshot from Amazon Mechanical Turk task list. The list presents a set of tasks, expiration date and intended reward. 13
2.3 Screenshot from FoldIt interface. 14
2.4 Crowdforge Framework: Splitting up and recombining complex human computation tasks based on map-reduce model, extracted from [Kit+11]. 15
2.5 BMPN Model: Use case 'Job Posting', extracted from [All16]. 18
2.6 A classical Dining philosophers problem represened as Petri Net, extracted from [Nac]. 21
2.7 Example of workflow net for processing complain, extracted from [VDAVHH04]. 25
2.8 Business artifacts represented as ER diagram specifying the vendor tasks, extracted from [BHS09]. 28
2.9 Expectation Maximization algorithm general workflow. 48

3.1 A simple actor popularity poll. 58
3.2 A profile for refinement of task t_{like}. 63
3.3 A refinement of node n_1, replaced by the profile for task t_{like} in Figure 3.2. 66
3.4 Record to record arithmetic operations. 69
3.5 Union of datasets. 70
3.6 Application of semantic rule R_1. 72
3.7 Application of semantic rule R_2. 73
3.8 Application of semantic rule R_3. 74
3.9 Application of semantic rule R_4. 75

3.10 Complex workflow execution. C_0 represents the initial configuration with data D_{in} allocated to node n_{int}. C_1 is the successor of C_0: worker u_1 is allocated to node n_2, and $t_2 = \lambda(n_2)$ is a complex task. C_3 depicts the configuration after refinement of node n_2 by a new workflow W_{t2} (shown in the grey rectangle). 77

4.1 Join operation Example. 85
4.2 A complex workflow. 93
4.3 Rewriting graph for the complex workflow shown in Figure 4.2. 93
4.4 An example showing ψ_{AE} constraints. 102
4.5 An example showing ψ_{EA} constraints. 102
4.6 Different scenario of a terminated run: with and without correct set of output. 103
4.7 Original workflow W(left). The new workflow is represented as W' after refinement of node n_1 in worklow W(right). 108
4.8 Symbolic Execution Tree. 109

5.1 Generating function. 127
5.2 The threshold values based on current estimate on consumed budget and fraction of task remaining at the beginning of a round. 132
5.3 A possible state for algorithm 2. 134
5.4 Comparison of cost vs. Accuracy. 138

6.1 A workflow in a smart city. 144
6.2 Exclusive fork. 146
6.3 Non-exclusive fork. 147
6.4 Synchronous execution. 147
6.5 Asynchronous example. 148
6.6 Join example. 148
6.7 Union example. 149
6.8 Corresponding phase example. 156
6.9 Workflow W with non-exclusive fork phase. 158
6.10 Workflow W' with a new phase p'_x replacing the phases p_2, p_3 of the workflow W. 158
6.11 Workflows with different orchestrations. 164

6.12 Distributions of workers accuracy(Pool of 50 workers). 165
6.13 Accuracy and cost comparison on low expertise. 167
6.14 Accuracy and cost comparison on Workflow 1. 168
6.15 Accuracy and cost comparison on Workflow 2. 168
6.16 Accuracy and cost comparison on Workflow 3. 169
6.17 Accuracy and cost comparison on Workflow 4. 169
6.18 Accuracy and cost comparison on Workflow 5. 170

A.1 Encoding of $inc(q, c_x, q')$ instruction. 203
A.2 Encoding of $dec(q, c_x, q', q'')$ instruction. 203
A.3 Encoding of Non-zero test followed by decrement. 204
A.4 Encoding of Zero test followed by state change. 205

LIST OF TABLES

2.1 An example of **CINEMA** database, extracted from [AHV95]. 30
2.2 JOIN query result. 38

4.1 Complexity of Termination ($EXPT$ stands for EXPTIME). 106
4.2 Complexity of Correctness ($EXPT$ stands for EXPTIME). 106

5.1 Datasets description. 135
5.2 Comparison of EM+aggregation (with Recall, specificity & task difficulty) w.r.t MV, D&S, GLAD, PMCRH, LFC, ZenCrowd on *Duck Identification* dataset 136
5.3 Comparison of EM+aggregation (with Recall, specificity & task difficulty) w.r.t MV, D&S, GLAD, PMCRH, LFC, ZenCrowd on *Product Identification* dataset 136
5.4 Comparison of EM+aggregation (with Recall, specificity & task difficulty) w.r.t MV, D&S, GLAD, PMCRH, LFC, ZenCrowd on *Sentiment Popularity* dataset 137
5.5 Running time(in seconds) of CrowdInc, MV and Static EM. 137

6.1 Evaluation Parameters. 166

PART I

Prologue

CHAPTER 1

INTRODUCTION

With the exponential generation of new data, organizations struggle with the information overload to manage data, design algorithms, and to extract knowledge. Almost 90% of the data that are present today are unstructured. It consists of images, videos, and natural language texts. It needs to be integrated, processed, stored, indexed, mined, tracked, and reported to meet the business needs of an organization in data-aware competitive markets. While the organizations rely on automated mechanisms powered by data management and artificial intelligence algorithms, complete automation is still decades far away. Also, some tasks are better performed by humans and are human-centric (for example, polling tasks, data collection, etc.). In these cases, humans can play a key role to analyze the data which requires natural intelligence.

Humans are good at understanding data in all forms text, image, and video, and can process, analyze data that are still hard for computers to apprehend. The widespread availability of the internet has paved the way to use the wisdom of the crowd to process the data. The term crowdsourcing is made of two words *crowd* and *sourcing* and the idea is to outsource the tasks to the human crowd to obtain data, ideas, and answers. A definition for crowdsourcing was proposed in [EAGLDG12].

Definition 1 (Crowdsourcing)**.** *Crowdsourcing is a type of participative online activity in which an individual, an institution, a nonprofit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task of variable complexity and modularity and in which the crowd should participate, bringing their work, money, knowledge **[and/or]** experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and use to their advantage that which the user has brought to the venture, whose form will depend on the type of activity undertaken.*

Crowdsourcing platforms such as Amazon Mechanical Turk (AMT) [Amta] is one of the most popular online marketplaces which promises to have millions of workers; other platforms include wirk [Wir], Figure Eight [Cro], 99designs [99d], Clickworker [Cli], etc. The workers receive incentives (money, goodies, motivation, ...) to realize the tasks at crowdsourcing platforms [Kuc+16]. The global crowdsourcing market was valued US $ 9, 519.53 million in 2018 and is expected to reach US $ 154, 835.74 million by 2027 and is growing at an average rate of 36.5% [Glo]. The availability of fast internet services has pushed the boundaries of these platforms to hire cheap and reliable workers throughout the world. The rapid growth of crowdsourcing introduces economical, legal, philosophical, and ethical issues related to the type of services, the nature of the task, the worker's activity, working environment, worker pay schemes, etc. [Sch13]. We do not take a position on ethical aspects. This thesis only concerns the technical aspects of crowdsourcing.

The crowdsourcing mechanism is simple and is shown in Figure 1.1. A client has a task and is willing to execute it at a crowdsourcing platform. The client submits the task to the platform. Then the crowdsourcing platform allocates the task to several workers. The workers execute the task using their expertise and beliefs and then return the results to the platform. The platform aggregates, processes the results, and in return workers receive the incentives. After the processing, the results are returned to the client by the platform.

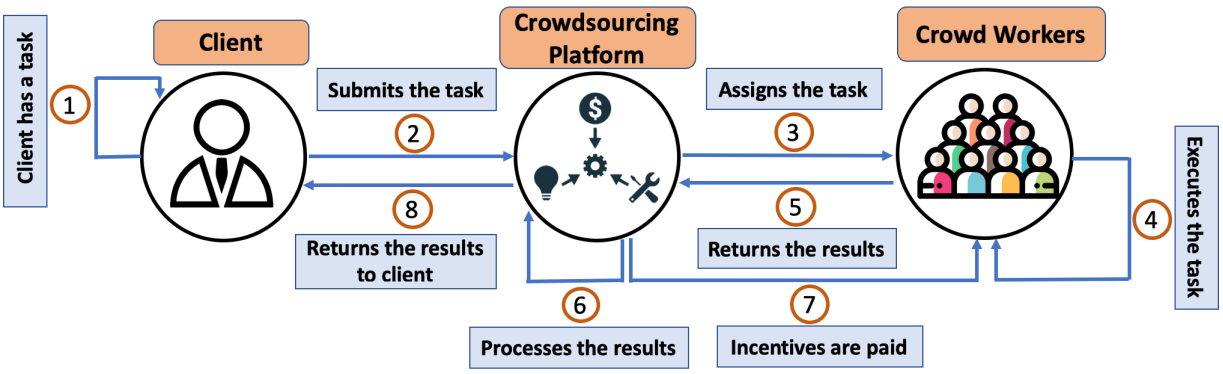


Figure 1.1 – Crowdsourcing mechanism.

Crowdsourcing has been applied in various domains. We list below several applications:

— AI - Training Data: Crowd workers are asked to tag images, videos with a particular label. The other tasks include speech recognition, audio tagging, and video analysis [HZS17; Esk+13].
— Text Processing: Humans expertise is used for entity resolution [BT06], sentiment analysis [Liu12], spam identification, product description, glossary & dictionaries, company profiles, blog articles and translations [KWD10].
— Surveys: Survey of a particular topic are easier to realize via crowdsourcing platforms which provide a wide variance of crowd workers within a few clicks [Beh+11].
— Search: Most search engine Google [Goo], Bing [Bin], Yahoo [Yah] require human workers to validate and improve the search results [Mor19; LGC15].
— Industry Solutions: The crowdsourcing platforms are used for online marketing, E-commerce, and advertisements [PCZ15]. For example, crowdsourcing is used to analyze the reviews which are hard for computers to process [Wu+15].
— Research: Projects such as FoldIt [Fol], a participatory science project where human workers contribute to solving puzzles of protein unfolding.
— Knowledge platforms: StackExchange [Sta], Quora [Quo] relies on human worker expertise to build collaborative knowledge intensive platforms.

Most of the current crowdsourcing platforms such as AMT allows the realization of a large batches of tasks. The task considered are mainly micro-tasks. Micro-tasks are small, quick, and independent tasks that require a small amount of time to complete (tagging an image, answering a simple boolean question, etc.). However, in real life, tasks are often complex and require several different skills of the workers (not only one as specific to micro-tasks). Consider a complex task: Gather insect images, judge the image quality, annotate the high-quality image using a pre-defined taxonomy, and then write a brief description for each image. Such intricate tasks are challenging. First, a model to define a complex task in terms of orchestrations of smaller tasks is required. It demands to specify the task as well as to orchestrate its realization. Orchestration is not only a question of scheduling micro-tasks: The data needs to be forwarded from one micro-task to the following ones. Also, the whole execution process should terminate with a correct set of outputs. The answers provided by the crowds are subjective and error-prone. To alleviate the problem and to maximize *accuracy*, generally, tasks are realized by several workers. Hence the answers from different sources need to be

aggregated. Additionally, a complex task comes with a pre-defined budget that allows hiring workers and rewarding them for the task realization. Some tasks may require a few answers to reach an agreement, while other intricate tasks (for instance, tasks calling for opinions or using worker's beliefs) may require more answers to forge an accurate result. It demands some mechanism to spend the budget in an optimized way enabling a trade-off between cost and accuracy. Hence, the complexity of deployment of the complex tasks along with the optimization of cost and accuracy correlates in complex ways. It is hence difficult to optimize the trade-offs between cost and accuracy, and at the same time guarantee progress and termination of a complex task distributed to crowd.

The objective of the thesis is to define techniques to deploy complex applications on top of conventional crowdsourcing platforms and to provide data centric algorithms optimizing cost and accuracy. We address these challenges by first defining formal models for complex workflows and then providing probabilistic models to handle the trade-off between cost and accuracy. We also provide a tool to check the termination and correctness properties of complex workflows.

1.1 Thesis Overview and Contribution

The existing crowdsourcing systems come with limited functionalities. The major limitation for such a system is the only support for micro-tasks execution. Irrespective of task intricacies, the crowdsourcing platforms call to shape the tasks as a batch of micro-tasks. The batches are handled independently with a simple pipeline. Each micro-task is extracted from the batch, affected to a worker in the crowd. The answer of the worker is added to an output batch. No information exchange between micro-tasks can occur in such a setting. One can only work with this kind of task deployment if tasks are independent work units, i.e. they can be realized in any order, and there is no data dependency between micro-tasks. However, real-world projects are complex and need several phases to be realized. One can also notice that many contexts (for instance text writing applications) call for data exchanges to reach the final objective of the overall process. Our goal is hence to build a human-powered data-driven system to realize complex tasks on top of a conventional crowdsourcing system.

The first contribution of the thesis is a model to specify and deploy complex tasks relying on existing crowdsourcing platforms. Complex tasks with pre-defined final objectives require step-wise processing of tasks in several phases. Each phase has an individual goal. As for example, a phase goal is to annotate a set of images as *clear* or *blur* while, the next phase considers only the *clear* images as input from the predecessor phase and is assigned the goal to *categorize* the image into particular taxonomy. The phases may have different prerequisites: some phases may require a general worker while others may require an expert one. The phases depend on each other and exchange data. We model complex tasks as workflow orchestrating sub-tasks. The orchestration as sub-tasks can be realized by workers, i.e., we consider higher-order answers in which a crowd worker does not realize a task by returning plain answers, but rather answer by returning a complex orchestration allowing to obtain the expected answer. We define a model called **complex workflows**, which is used to orchestrate different parts of complex tasks. In addition to workflow-based tasks-coordination, we allow for the definition of worker's skills, input data (constraints on possible inputs), and higher-order answers, addressed as rewriting rules.

Orchestrations of complex tasks and higher-order answers come with challenges. Higher order allows answers of workers defining *how to obtain answers* rather than crisp data. Given input data provided by the client, Complex workflow use the knowledge and skills of crowd workers to complete a complex task i.e. realize successive phases of a complex process, hire workers, collect answers, process data, and return the final result to client. However, a workflow may never terminate. It can be due to particular data input that cannot be processed properly by the workers or to infinite recursive schemes appearing during the execution, to deadlocked situations due to missing worker competencies. One challenge is to decide whether a complex orchestration of tasks *always* terminates for a particular input dataset, or for all input datasets that meet some constraints (such as constraint on domains used). In practice, one does not want wrong data or wrong choices of workers to block a task realization, but this can nevertheless occur. First the question is whether the orchestration *can* terminate: if this is not the case, then the task definition should be considered ill-formed. Second challenge is to guarantee that the workflow gives correct output at each of the phases and at the end produces the desired output data as required by the client. We address the question of existential termination (at least one execution of the workflow produces a final result) and of universal termination (all executions of the workflow terminate).

The second result of this thesis is that existential termination is undecidable in general and on the other hand, universal termination is decidable. We then show that, when restricting the number of refinements of tasks, existential termination becomes decidable. Then, termination boils down to the satisfaction of data dependencies between inputs and outputs of each step in a finite set of runs of the workflow. These dependencies can be expressed in a decidable fragment of first-order (FO) formula. We address termination as a question of satisfiability of a series of weakest preconditions for a run. We show that many fragments of FO are closed under the calculus of weakest precondition. This allows for the construction of algorithms to check the realizability of a particular run from the initial configuration of the workflow to a final successful one. This algorithm is then used to prove the decidability of universal termination and existential termination for specifications with bounded recursion. We also consider the question of correctness that holds if a workflow terminates and the produced output meets constraints on data provided by the client. Again, if output constraints are expressed in decidable fragments FO, correctness is decidable.

Workflow provides an efficient way to synchronize complex tasks in the form of different phases to achieve business goals. Nevertheless, a workflow alone is not enough to fulfill important non-functional requirements such as the accuracy of the data produced, and the cost for the realization of the workflow. As a third contribution of the thesis, we consider a divide and conquer philosophy and focus on a cost and accuracy trade-off for tasks achieved by replication of a single micro-task. We address trust in this setting by replicating the considered micro-task as much as needed, i.e. we hire new workers from the crowd to get new answers, and aggregate the collected answer as long as a certain confidence level is not reached and as long as the overall budget for this task allows it. To alleviate the problem, we allocate each of the tasks to a set of workers and the goal is to somehow combine the given answer set to reach a final consensus. The simplest approach for aggregation is Majority voting. We however show that majority voting is not a good approach both in terms of cost and accuracy. Similarly, we show that statically allocating as many workers as allowed by a budget to tag their subset of record from an input dataset is not a good way to address cost. We propose an aggregation technique, that considers answers returned by workers, but also various hidden variables such as task difficulty and workers accuracy. We define an expectation maximization (EM) based algorithm to derive the value of hidden variables and compute final aggregated answers with a confidence score. We handle cost

by a dynamic allocation policy that hires new workers to tag records with a low confidence score. We demonstrate that the EM aggregation combined with the dynamic worker allocation algorithm outperforms the existing approaches both in terms of cost and accuracy.

The fourth contribution of the thesis builds on the algorithm to optimize cost and accuracy for a single-phase and extends the results to complex workflows. Addressing cost and accuracy for orchestrations of tasks is more complex than for replication of a single micro-task. First, there exist two possible approaches to execute a complex workflow, namely a synchronous approach, and an asynchronous one. In synchronous execution, after completion of tasks tagged with records appearing in a dataset. (e.g. after tagging all images provided), all records move to the next phase(s). On the other hand in asynchronous execution of the workflow, records are forwarded to the next phase(s) of a workflow as soon as they are considered as processed. We show that modes of execution require different trade-off mechanisms in terms of cost and accuracy. We propose dynamic worker hiring policies that build on the EM algorithm shown before. This allows for the optimization of cost and accuracy in the context of synchronous and asynchronous workflow realization. We show the pertinence of the proposed algorithms on samples composed of different workflows, pools of workers with different expertise, and different characteristics of data.

The last contribution of this thesis is a tool to verify the specification of the complex workflow. The tools take as input a specification of workflow in terms of nodes, tasks, workflow, workers, workers skills, task constraints, and data. The tool derives the minimum condition required to process the input data at each node and check the satisfiability of the formulas. The tool checks satisfiability of the weakest preconditions that are necessary to terminate execution, starting from the final nodes of the workflow, and ending on the initial node. If all weakest preconditions met when moving backward from a final node are satisfiable, then the followed path witnesses a terminating run. We use this technique to prove universal termination and existential termination of non-recursive specifications.

1.2 Outline

The thesis is organized as follows. The state of the art (Chapter 2) introduces the models and notations needed in the document and presents the works related to the contents of the thesis. We do a brief survey on crowdsourcing, marketplaces, and current mechanisms followed by industry. We then present a detailed study on different works on the orchestration of tasks and languages; particularly we address process-centric approaches, artifact-centric approaches, Petri net-based models, active XML, and UML-based languages. Data is the core of our model. We present different ways to handle data in the context of our thesis. We recall the basics of first-order logic, datalog, SQL and weakest preconditions. We then survey the literature on cost and quality trade-offs in crowdsourcing.

Chapter 3 presents the formal semantics of a model designed to specify complex tasks. We define components for a workflow: client, tasks, skills, task prerequisite, and data. We then introduce the formal semantics and rules for the workflow. We also introduce higher-order constructs and show on use cases how these higher-order answers can be used to capture the knowledge of crowd workers.

Chapter 4 addresses the questions of termination and correctness of complex workflows. We show that existential termination is undecidable in general and on the other hand, universal termination is decidable. We also show that by restricting the number of refinements of tasks, existential termination becomes decidable. This restricts the workflow to an orchestration of a bounded number of tasks. We give an algorithm to check termination of a complex workflow with bounded recursion based on the derivation of the weakest precondition. Next we consider the question of correctness that holds if a workflow terminates and the produced output meets constraints on data provided by the client. Again, if output constraints are expressed in decidable fragments FO, correctness is decidable. We analyze the complexity of decidable termination and correctness problems of the workflow. We also show a tool named *CrowdPlex*, designed to check the termination of a complex workflow. We explain the building blocks used by the tool to check termination.

In Chapter 5, we introduce our cost and accuracy trade-off model for crowdsourcing of a single task. We first study the aggregation problem: infer the correct final answer from a given set of answers provided by workers. We consider the two key factors *difficulty* of a task, and *expertise* of workers, expressed in terms of *recall* and *specificity* to

model the task answer-ability by a worker. Aggregation is based on expectation maximization algorithm which jointly estimates the answers, the difficulty of tasks, and expertise of workers. We propose an algorithm called *CrowdInc*, an incremental labeling technique that optimizes the overall cost to collect answers and aggregate them. The algorithm implements a worker allocation policy that takes decisions from a dynamic threshold computed at each round, which helps to achieve a good trade-off between cost and accuracy. We evaluate the algorithm on real datasets to validate our approach. We show that our aggregation approach outperforms the existing state-of-the-art techniques. We also show that the incremental crowdsourcing technique achieves the same accuracy as EM with the static allocation of workers, better accuracy than majority voting, and in both cases at considerably lower costs.

Chapter 6 extends the algorithm of Chapter 5 to the complex workflow setting. A task in a workflow can be realized in two ways, i.e. synchronous and asynchronous execution. We add the notion of *phase* to our complex workflow model and revisit its semantics. We then study dynamic worker allocation for synchronous and asynchronous workflows realization. At last, we compare our results with the conventional crowdsourcing approaches and find the proposed technique achieves at least same accuracy with a reduced budget. We present the cost and accuracy analysis and results in different configurations.

Finally, we conclude the thesis in Chapter 7 with a summary of achieved results, discussion, possible improvements and we define future research directions.

CHAPTER 2

STATE OF THE ART

We organize the chapter as follows: we first introduce crowdsourcing marketplaces and challenges in context to the thesis. We then discuss the orchestration of tasks, languages, and data aspects. In the end, we brief about the cost and quality assurances in crowdsourcing.

2.1 Crowdsourcing Marketplaces

With the advent of web 2.0, a large number of platforms have emerged in the last decade that uses the power of the crowd to solve problems. In this thesis, we particularly focus on the platforms which use human as data processors. For example, data can be a set of images and the task is to annotate each of the images with a tag chosen from a particular taxonomy. Here, human acts as a data processor that takes some data as input and return the processed output, i.e. tagged data.

There exists numerous crowdsourcing marketplaces where requesters post their tasks and workers realize tasks in exchange for some incentives. The major platform includes Amazon Mechanical Turk [Amta], wirk [Wir], Figure Eight [Fig], 99designs [99d] and Clickworker [Cli]. Consider the most well-known platform, AMT. The platform realizes almost five million tasks each year with an incentive between one and ten cents per task [Ipe10]. Amazon borrows its name from a chess-playing machine named *Mechanical Turk* constructed by *Wolfgang von Kempelen* in the late 18th century that toured Europe beating both *Napoleon Bonaparte* and *Benjamin Franklin*. The machine was a hoax and a mechanical illusion that allowed human chess masters to hide inside to operate the machine. Artificial intelligence promises to solve problems that require intelligence and natural understanding. However, in present times, such automation is not possible for all types of tasks and still requires human intelligence. In a similar vein and perhaps to banter, amazon coined the platform AMT as *artificial artificial intelli-*

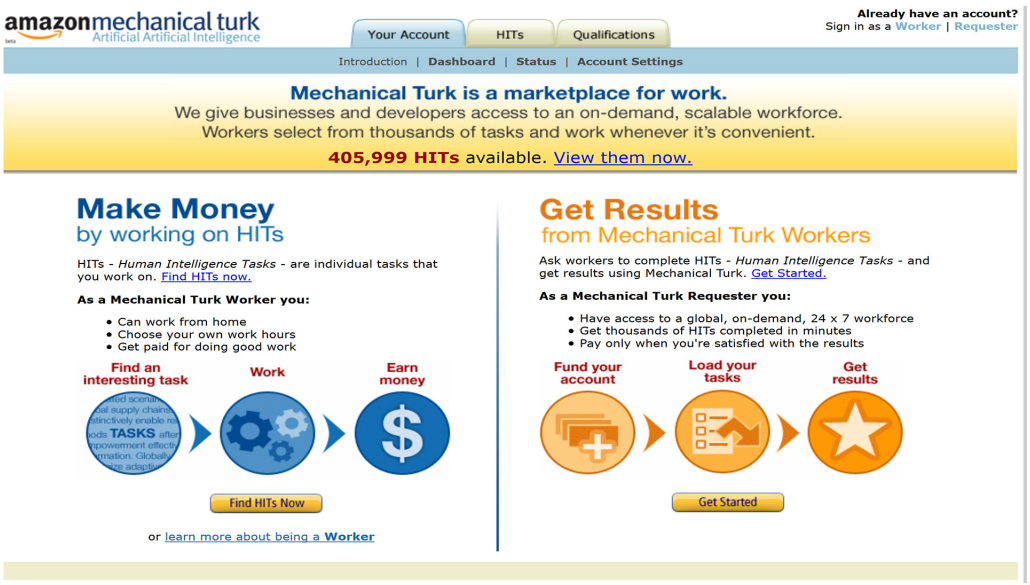


Figure 2.1 – Screenshot from Amazon Mechanical Turk.

gence as a process to outsource some parts of the program to humans. The online service uses remote human workers to hide behind the machine to help requesters to realize tasks that are still hard for computers to solve. Most of the workers at AMT come from the United States and India [Ros+10].

The blueprint and design of each platform vary, but in general, follow similar steps. Consider the example of AMT. The platform stakeholders are requesters and crowd workers, as depicted in Figure 2.1. A requester publishes his tasks on the AMT platform and decides how much to pay to each worker for each assignment. The tasks at AMT are called Human Intelligence Tasks (HIT) and generally require a very small amount of time to realize. The minimum fee is $0.01 per assignment. The MTurk platform takes 20% fee on the total reward [Amtb]. A requester may also grant some bonus after the realization of the task based on the performance of the worker. The tasks are published on the platform with descriptions as requester name, expiration date, the time allotted, and rewards as depicted in Figure 2.2. For some of the tasks, the requester can also require a qualification test. The crowd workers visualize the tasks on the platform and bid for the tasks. In the end, the workers who realize the task in the given time frame are awarded incentives and extra bonuses.

Figure Eight (formerly known as Dolores Lab, CrowdFlower) [Cro] is another popular platform. It comes with a more user-friendly interface, its crowd, and its built-in

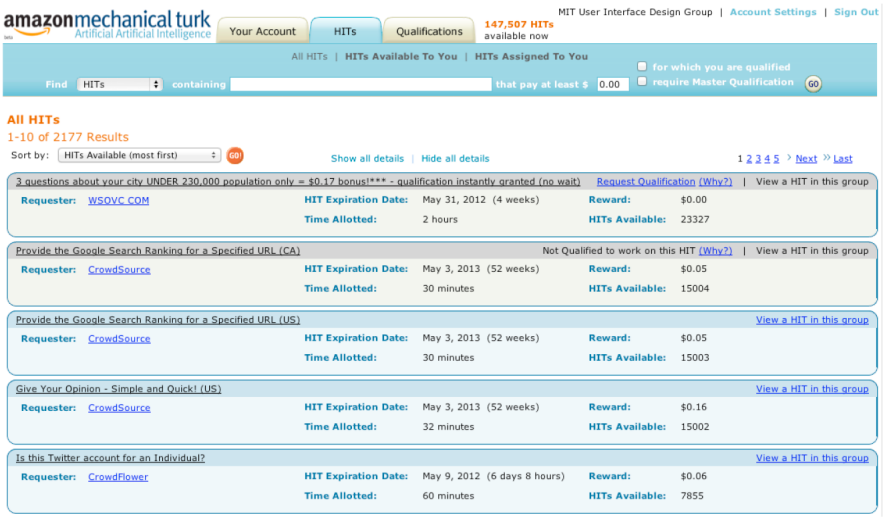


Figure 2.2 – Screenshot from Amazon Mechanical Turk task list. The list presents a set of tasks, expiration date and intended reward.

features. The platform is more focused to provide large volumes of world-class training data to AI engines. Wirk [Wir], a subsidiary of Foule factory is a french platform that provides optimized resources, monitor and steering, service audits, workflow design along a community of 50,000 freelancers. This thesis is realized in the context of HEADWORK[1] ANR project with active participation and support of the wirk platform. Other platforms such as Freelancer [Fre] and Upwork [Upw] provide an expert workforce that receives higher incentives.

Apart from commercial crowdsourcing platforms, there exist other academic and community platforms. FoldIt [Fol], is a popular citizen science experimental research project developed by the University of Washington. It provides an online puzzle video game about protein unfolding and has around 240000 registered players. The objective of the game is to fold the structures of selected proteins as perfectly as possible using the provided tools as depicted in Figure 2.3. The platform gained its popularity when a non-expert worker discovered a new protein unfolding and led to several scientific publications. The paper published in nature [Kha+11; Coo+10] credited FoldIt's 57000 players that provided useful results that matched or even outperformed algorithmically computed solutions.

1. This work was supported by the Headwork ANR project (ANR-16-CE23-0015)

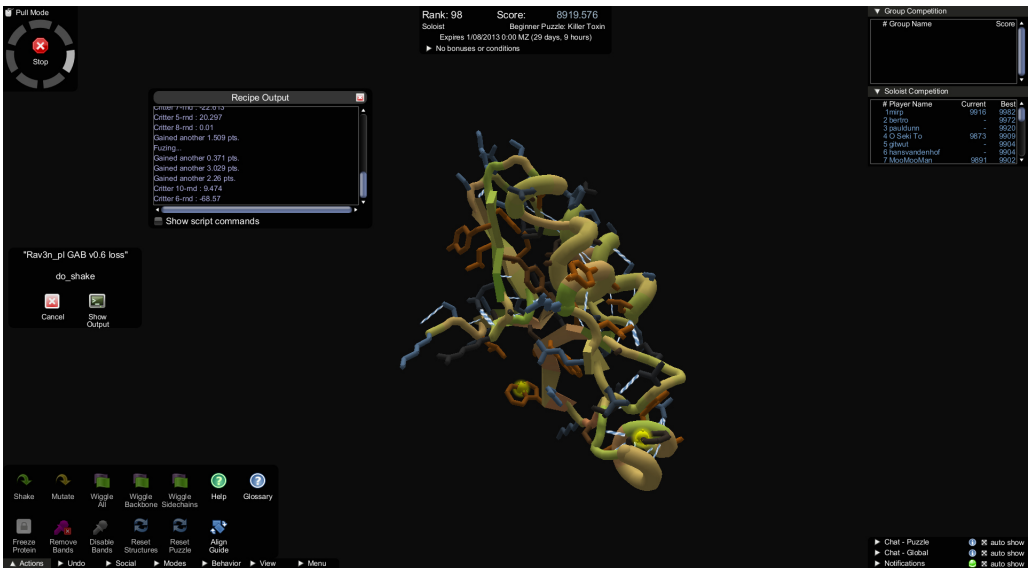


Figure 2.3 – Screenshot from FoldIt interface.

2.2 Crowdsourcing Space

Crowdsourcing has increased the number of problems that are solvable in a semi-automated way, even if they are still inherently hard to solve for a computer, at least in a foreseeable future. However, there exist several challenges to give its full potential to crowdsourcing. A crowdsourcing system builds on many paradigms. Trust and management of workers directly impact the execution of tasks in crowdsourcing systems. Several workers have considered trust in crowdsourcing [Yu+12]. Crowd trust [YWL15] is a context-aware trust model for worker selection in crowdsourcing environments. Skills mapping is another paradigm that is widely studied in crowdsourcing and includes techniques such as hierarchical taxonomy-based skill mapping in form of tree [MGAM16], self-assessment-based worker allocation [Gad+17]. Task design of the crowdsourcing system is another dimension that directly impacts the understanding and execution of the task [ZLH11; Fin+13; BW18]. Another interesting paradigm is query processing. CrowdDB [Fra+11] proposes a query processor which takes human input via crowdsourcing to process queries that neither search engines nor database system can answer. A more sophisticated query processer is Deco [Par+12] which proposes a database system for declarative crowdsourcing. Declarative crowdsourcing hides the complexities to retrieve the data. The user is only required to submit a SQL-like query

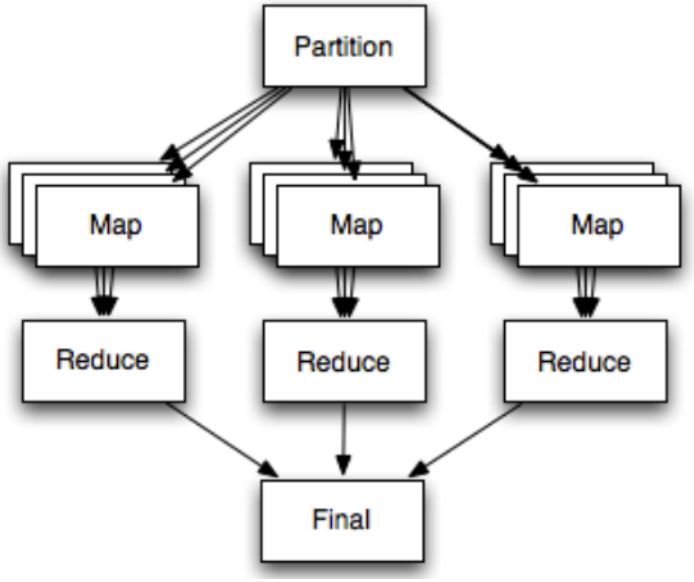


Figure 2.4 – Crowdforge Framework: Splitting up and recombining complex human computation tasks based on map-reduce model, extracted from [Kit+11].

and the platform compiles the query, generates the execution plan, and gets answers from the crowd. It relieves the user of the burden of dealing with the crowd and returns data as an output of a submitted SQL-like query. Workers contributing to a task receive incentives [ZLM14; KSK16; Dan+18]. The more workers contribute to a task, the more reliable the result. However, as incentives are paid, tasks usually come with a limited budget, which calls for trade-offs between the cost of a task realization, and the achieved accuracy [Dan+18]. We discuss in detail the cost-accuracy tradeoff in Section 2.5. There exists a large number of comprehensive surveys describing aspects of crowdsourcing [QB11; CCAY16; Mao+17; GM+16; Li+16a]. In this thesis, we do not address all challenges related to crowdsourcing and focus on the execution of a complex task which is the primary topic of the thesis.

Existing crowdsourcing platforms are mainly systems that distribute collections of a replicated micro-tasks which are simple and independent. Most of the work in academics and industry proposes solutions for data acquisition and management mainly at the level of micro-tasks [GM+16]. The realization of complex tasks on crowdsourcing platforms is a recent topic. Complex tasks require coordination of various small tasks and are typically not supported by existing platforms. We present here the studies that focus on complex tasks.

Crowdforge [Kit+11] is an interesting work that uses the Map-Reduce technique to solve complex tasks. The orchestration of the task is depicted in Figure 2.4. It provides a graphical web interface to decompose complex tasks into sub-tasks, establish relations, workflows, and dependencies among the sub-tasks. The basic understanding of a programming language (Python) along with a predefined way to decompose the complex task into sub-tasks is the major limitation of the platform. The prototype was designed for task designers and does not give to crowd workers the power to decompose a task into sub-tasks. Turkit [Lit+09] is another work that uses a crash and rerun programming model for crowdsourcing applications. The tool allows to write imperative programs and calls the crowdsourcing platform as a function in an iterative fashion. The author claims the fault-tolerant model can be widely applied to various crowdsourcing tasks. Here, the requester must know the way to divide the tasks into sub-tasks. [KCH12] proposes the Turkomatic tool which works on the principle of Price, Divide and Solve (PDS). The tool decomposes a complex task with the help of human workers while the requester can watch the decomposition workflow. It also allows the client to intervene during the execution of tasks. In such approaches, the requester is required to monitor the whole workflow. Such scenarios are often not suited as workflow monitoring requires a lot of patience and time by the task requester. A conceptual meta-model based on the combination of PDS and hierarchical state machines is presented by [Zhe+16]. The tasks are defined with states $S = \{Initial, Decomposition, Judge, Solve, etc.\}$. The paper formalizes complex crowdsourcing tasks as sequences of states. The model lacks the means to orchestrate sub-tasks in parallel. We will show in the next chapters that this can be easily achieved by introducing concurrency in the workflows. The author does not comment on the formal termination and correctness of the state machines which can lead to undesirable results. Soylent [Ber+15] embeds the crowd workforce directly into its user interface. It focuses on complex word processing task which requires multiple levels of conceptual and pragmatic activity. The interface enables writers to call crowd workers on the AMT platform to realize various tasks as shortening, proofreading, etc. The author introduces the concept of Find-Fix-Verify, a crowd programming pattern that splits the tasks into three steps of identification, generation, and verification. The author [SC+15] presents a graphical framework named CrowdWON for complex tasks based on an adaptive workflow net. A graphical net with a deadline mechanism is presented to design, describe, and visualize the flow of tasks at crowdsourcing platforms. The article

articulates well the workflow and time constraints in the design of tasks. However, similar to former models the study of formal properties as termination and correctness is not investigated. The work in [Kit+12] proposes the *CrowdWeaver* toolkit to visually manage complex crowd work. The tool acts as a mental model designing a task, integrating human and machine, templates, incentive distribution, tracking, and is built on top of CrowdFlower. The tool architecture is built for the management of micro-tasks on the crowd platform. Authors in [Tra+15] state composite tasks are poorly supported by crowdsourcing platforms and propose *Crowd Computer* that allows one to program custom logic for individual and structured tasks. The model is based on a business process model and notation that provides process logic at a level of abstraction mostly suitable for the specification and coordination of tasks.

Models for complex tasks call for complex mechanisms to orchestrate sub-tasks. Many models proposed in the literature are not formal enough. This is a clear limitation to address verification of properties (correctness, termination, ...). Another gap in many models is the consideration of data as a second-class citizen. Most of the models are process-driven rather than data driven. In this thesis, we propose high-level coordination mechanisms for crowdsourced tasks that include data. We also provide means to verify the termination and correctness of a crowdsourcing system.

2.3 Orchestration of Tasks and Languages

A simple and independent task takes input and generates some output. For example, a task can take a sentence as an input, and return an output which is a tag for the sentence from a predefined category: { *Positive, Negative, Neutral* }. However, in real life, we often encounter tasks that are not simple tagging tasks and are rather complex. Such tasks need to be decomposed in smaller steps, that have to be orchestrated to produce the desired output. Such orchestrations are called processes. Thomas Davenport defined a *business process* in his book Process innovation [Dav93] as *a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. Taking a process approach implies adopting the customer's point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers.*

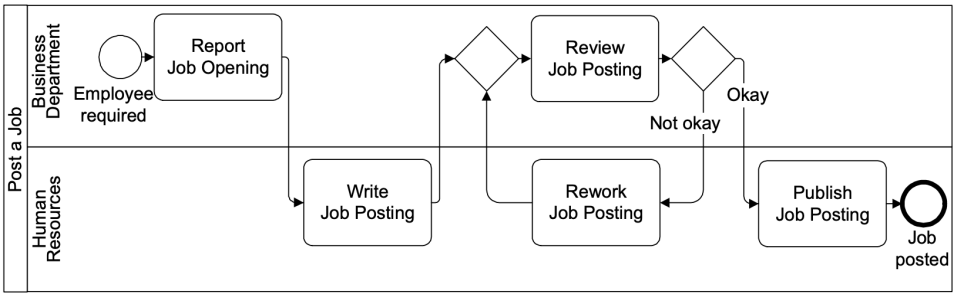


Figure 2.5 – BMPN Model: Use case 'Job Posting', extracted from [All16].

In a similar context, Weske [Wes12] defines a business process as "*a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal*". Usually, a business process is an explicit graphical representation of activities with dependencies and constraints. Using these models improve control, and should increase the efficiency of the system both in terms of cost and quality. Business process modeling helps to represent day-to-day business activities utilizing a model.

2.3.1 Process Centric Approaches

Several languages exist to represent business process models based on the traditional process-centric approach. Business Process Modeling Notation (BPMN) is one of the most popular among them. It is standardized by Object Management Group [2]. BPMN is a notation for business process modeling that uses Business Process Diagrams (BPD) to graphically represent a business process. BPD uses flowcharts similar to activity diagrams in the Unified Modeling Language(UML) [Whi04]. We present an example from [All16], that showcases the representation of a Job Posting in the BPMN model. The model is represented in Figure 2.5.

The case study is the following: A company wants to post a job opening based on some requirements. The process "Post a Job" involves two departments: *Business Department* and *Human Resources*. The business department reports job openings. The human resource department then writes the job posting. The written document is then sent for the review to business department. The business department has two options. If the written document is okay then they send an affirmation to *Publish* the

2. **https://www.omg.org/**.

Job to human resource department. In the second case, if it is not okay, the business department tells to rework on the draft. The iteration can happen multiple times. In the end, when the document is ready, it is published by the human resource department.

Web Service Business Process Execution Language (WS-BPEL), also known as BPEL is a language, tailored for specifying business processes with web services [Jor+07]. It follows XML notation and is standardized by the Organization for the Advancement of Structured Information Standards (OASIS) [OAS07]. BPEL facilitates sending and receiving messages along with facilitates some plug-in models that allow writing expressions and queries. As compared to BPMN, BPEL does not provide the graphical front end to show the process descriptions. Yet Another Workflow Language (YWL) is another language based on workflow patterns. YAWL uses XML to define and manipulate data, to monitor and control workflow. YAWL and BPEL models are very close and often considered as an alternative to each other. However, BPEL is more popular due to standardization, community, and business supports. The advantage of YAWL over BPEL is that YAWL supports processes that require a human contribution.

Note that the majority of process-based models focus on the orchestration of tasks and represent this orchestration as a workflow diagram. However, the business process model often lacks information on the data aspects of a process. Consider the "Job Posting" case, here we get a basic idea about the control and execution flow of the overall tasks using the BPMN diagram, but it lacks data details. For example *What are the attributes of the Job posting advertisements? On what basis and features, business departments decide to rework the project? What are new transformations on the draft as compared to the previous version?* Clearly, business processes manipulate data, and integrating data to these formalisms is essential. These models hence need to provide data description, and means to specify how data is manipulated at each task. These models also lack dynamic orchestrations, i.e. workflows are usually statically defined and cannot be refined at run-time to improve a process [KCM06].

UML based languages

Unified Modeling Language (UML) [Fow04; Boo05] is an object-oriented visual language that has emerged as an industry standard for the representation of artifacts. It has wide applications such as design, specification, visualization, documentation for business modeling, etc. It describes any type of system through an object-oriented approach and can represent the system with static as well as dynamic behaviors. UML

includes five categories of maps. *Use case diagram* displays use case, actors, and their relationship between them. *State diagram* includes class diagram, object diagram and packet diagram. *Behavior diagram* consists of activity diagrams and models system's dynamical aspects. *Interactive diagram* describes interactive relationship among objects. *Implementation diagram* includes component diagrams. Note that UML diagrams present the basic association of activities but hide data. Class diagrams provide clear and well-adopted representations of data, but dynamic diagrams such as use cases usually do not address data manipulation. BPMN, BPEL, activity diagrams all focus on tasks and dependencies among them. Formally speaking, these features are well captured by a single formalism, namely Petri nets.

Petri nets based Models

Petri nets [Pet62] are formal language to model and analyze discrete event systems. Several formal models based on Petri nets are proposed for the orchestration of tasks. A Petri net is a directed bipartite graph and consists of places, transitions, and arcs. Places symbolize resources or states and transitions symbolize actions. Places may hold a discrete number of tokens, symbolizing available resources and the control flow of a system. Arcs connect places to transitions and transitions to places, and define a flow of tokens consumed or produced by a transition.

Definition 2 (Petri Nets)**.** *A Petri net is a tuple* $N = \langle P, T, F, W, m_0 \rangle$*, where*

- P *is a finite set of places*
- T *is a finite set of transitions*
- *The places* P *and transitions* T *are disjoint* $(P \cap T = \emptyset)$
- $F \subset (P \times T) \cup (T \times P)$ *is the flow relation*
- $m_0 : P \to \mathbb{N}$ *is the initial marking representing the initial distribution of tokens*
- $W : ((P \times T) \cup (T \times P)) \to \mathbb{N}$ *an arc weight mapping (where* $\forall f \notin F, W(f) = 0$ *and* $\forall f \in F, W(f) > 0$*)*

The places P and transitions T are disjoint $(P \cap T = \emptyset)$. A place p is called an *input place* of transition t if there exists an arc from p to t. Place p is called an *output place* of transition t if there exists an arc from t to p. Arc F denotes the flow relation. A *marking* is a map $m \to \mathbb{N}$ that assigns a natural number to each place of the net, representing the number of tokens held by that place at a given instant. $m(p)$ denotes the number of tokens that a marking m associates to a place p. At any time a place p

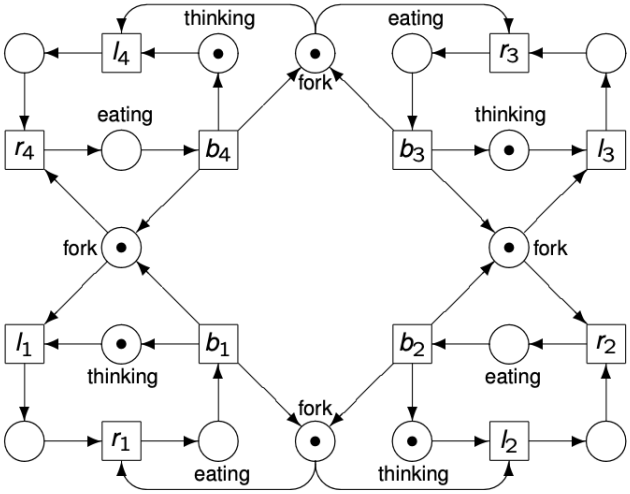


Figure 2.6 – A classical Dining philosophers problem represened as Petri Net, extracted from [Nac].

contains zero or more tokens, usually drawn as black dots. The semantics of a Petri net is defined in terms of transitions firing from markings. Intuitively, a transition can fire from a marking when a sufficient number of tokens is held at its input places. Firing the transition consumes (removes) tokens from upstream places and produces (creates) tokens in its downstream places. W indicates the number of tokens to be consumed during firing and produced after the firing. When $W(p,t) \leq 1$ and $W(t,p) \leq 1,\ \forall p \in P$ and $t \in T$, a net is called a basic Petri net.

An example of a Petri net representing classical dining philosophers is shown in Figure 2.6. Places are represented as circles, transitions by rectangles, arcs by arrows, and tokens by dots. Places and transitions are labeled to indicate the conditions and actions. Markings are represented by a number of tokens in the places, for example, $m(thinking) = 1$.

Definition 3 (preset and postset)**.** *The preset of a given transition* t *is the set of input places of* $t : \bullet t \triangleq \{(p|\langle p,t\rangle \in F\}$ *and postset of a transition* t *is the set of its output places* $t : t\bullet \triangleq \{(p|\langle t,p\rangle \in F\}$*.*

The notations can be also used for places: for each place, $p \in P$, $\bullet p$ denotes the set of transitions that may produce tokens in p and $p\bullet$ denotes the set of transitions that may consume tokens from p.

Semantics

The semantics of Petri net is defined in terms of transitions *firing*. Each firing consumes and produces tokens. Firing of a transition models the execution of an event corresponding to t. Steps in Petri nets semantics are simply moves from one marking to the next one. The system starts in an initial marking denoted by m_0. The semantics of the Petri nets model is formally defined as follows.

Definition 4 (enabledness)**.** *A transition t is enabled by a marking m iff $\forall p \in \bullet t, m(p) \geq W(p,t)$.*

A transition t is enabled if each input place p of t is marked with at least $W(p,t)$ tokens, where $W(p,t)$ is the weight of the arc from p to t. A transition t can fire only when it is enabled. When a transition is fired, it takes the tokens from the input places and then distributes the tokens to output places. The firing of an enabled transition is defined as follows.

Definition 5 (firing)**.** *Firing an enabled transition t from a marking m yields a new marking m' such that*

$$\forall p \in P \;\; m'(p) = m(p) - W(p,t) + W(t,p)$$

A firing of an enabled transition t consumes $W(p,t)$ tokens from each input place p of t and produces $W(t,p)$ tokens to each output place p of t. Here, $W(t,p)$ denotes the weight of the arc from t to p. We write $m \xrightarrow{t} m'$ that states firing t from a marking m produces a marking m'. A *firing sequence* for a Petri net N with initial marking m_0 is a sequence of transitions $\delta = \langle t_1.t_2.\ldots.t_n \rangle$ such that $m_0 \xrightarrow{t_1} m_1 \wedge \cdots \wedge m_{n-1} \xrightarrow{t_n} m_n$ or simply $m_0 \xrightarrow{\delta} m_n$.

Petri nets have an interesting expressive power. First, they can be used to represent some classes of concurrent systems. Second, they can represent systems with infinite state space: indeed, the number of tokens in markings is not a priori bounded, even for basic Petri nets. Petri nets systems can be used for the analysis of many properties associated with concurrent systems. We illustrate some of these properties.

Reachability. The *reachability* problem for Petri nets consists of deciding, given a Petri nets N, m_0, and a marking m of N, if there exists a *firing sequence* such that m can be reached from m_o. The set of all possible marking reachable from m_0 in a net N is denoted as $Reach(m_0)$. Reachability problem for Petri nets is decidable [May81] and was recently shown *non-elementary* [Cze+19]. Reachability property of Petri net is used to find the erroneous state in a concurrent systems.

Boundedness. A Petri net is said to be $k - bounded$ or simply *bounded* if the number of tokens in each place does not exceed a finite number k for any marking reachable from m_0, i.e. $m(p) \leq k$ for every place p and every marking $m \in Reach(m_0)$. Karp and Miller proved that boundedness is decidable [KM69]. The boundedness property can be used to model systems with limited resources. For example, Petri nets models can be used to represent buffers in simple production lines [Rec+03]. By verifying the net is *bounded*, it is guaranteed that no overflows will take place in buffers, irrespective of the firing sequence taken.

Liveness. A Petri net is *live* if every transition can always occur again. Precisely, if for every reachable marking m and every transition t, there exists a firing sequence $m \xrightarrow{\delta} m'$ such that m' enables t. Live Petri net guarantees deadlock free operation irrespective of the firing sequence chosen. The liveness problem is decidable [Hac76]. Liveness property helps to model concurrent systems where resources are shared.

Petri nets are widely studied and applied in various domains such as modeling communication protocols, manufacturing, hardware design, multiprocessor systems, railway networks, etc. [Mur89]. Many variants of Petri nets have been proposed to model such systems. A time Petri net is defined as $TP = \langle P, T, F, W, m_0, I_s \rangle$ where P, T, F, W and m_0 are places, transitions, arcs, and an initial marking $I_s : T \to \mathsf{I}^+$ is a function that associates a static time interval to transitions. This model allows for the specification of time elapsing between transition firings. Its semantics is hence a timed transition system. Such models are interesting to define systems with real-time constraints such as metros or trains. We do not detail this model here, and refer [Mer74] for more details. Stochastic Petri nets (SPN) [Mol82] incorporates randomness in Petri nets. Firing times of transitions i.e. the time a transition has to wait before its firing once enabled, follow exponential distributions. The stochastic behavior of bounded stochastic nets can be brought back to that of Markov chains and is analyzed in [BK98]. Stochastic time Petri nets [CGV08] extends SPN. They allow for the specification of time Petri nets where firing times are specified with intervals as in [Mer74] but these intervals are equipped

with a probability distribution that are not necessarily exponential laws. This model is very expressive, and cannot be analyzed as a finite Markov chain. However, [Hor+12] has shown transient analysis techniques for this model.

Petri net model allows multiple tokens in the system and provides richer semantics to model several processes competing for the resources for the performance of a given task. Tokens in the Petri nets represent objects and resources. However, one often needs to represent the attributes of these objects. For example, if *hardware* is represented as a token in Petri net, then we may want to represent attributes such as hardware id, manufacture name, manufacture year, etc. These attributes cannot be represented as a token in classical Petri nets. For modeling such systems, coloured Petri nets models have been proposed. Coloured Petri nets [Jen89; dFM18] allow tokens to have a data value attached to them and the data value is called the token colour. Adding colours to tokens and guards to transitions give huge expressive power to nets. Coloured Petri nets are Turing-powerful. There also exists some other data-aware variants of Petri nets [Leo+14; Laz+08; FLM19], nets are decorated with global variables. Transitions firings are guarded by formulas on variables, and execution of transitions updates variables values. Structured Data nets (SDN) [BHM16] is a variant of Petri nets where tokens are structured documents (XML) and transitions transform data. SDN considers a token as a piece of information that either belongs to a database associated with the system or is attached to some ongoing transactions. Transitions are guarded by boolean queries that evaluate the truth value of some pattern matching constraint. Their execution results in some rewriting specified again as a query. Unsurprisingly, this model is Turing-powerful. Under some restrictions on the allowed shape of data, properties such as termination, soundness are decidable. These models can be used to model business processes with data.

Workflow nets

Workflow nets [Aal98] is a subset of Petri nets to model and analyze workflows such as business processes. They allow parallel, sequential execution of tasks, fork, and join operations to create or merge a finite number of parallel threads. Tasks are represented by transitions. Workflow nets mainly deal with the control part of business processes, and data is not central to this model.

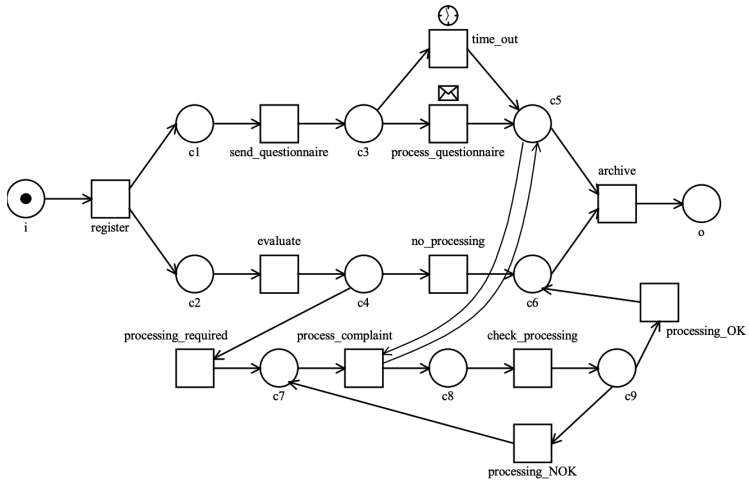


Figure 2.7 – Example of workflow net for processing complain, extracted from [VDAVHH04].

Definition 6 (Workflow Nets [Aal98])**.** *A petri net* $PN = \langle P, T, F \rangle$ *is a Workflow net (WF-net) if and only if :*

- *PN has two special places:* $i \in P$ *and* $o \in P$
- *Place* i *is a source place:* $\bullet\, i = \emptyset$
- *Place* o *is a sink place:* $o \bullet = \emptyset$
- *Every node* $x \in P \cup T$ *is on a path from* i *to* o

Places in the set P are called conditions, transitions in the set T are called tasks. The WF-net have one input place and one output place. These places indicate starting and ending state of the process. The semantic of WF-net follow the standard Petri nets semantics. A WF-net model for processing complaint is shown in Figure 2.7 [VDAVHH04]. First, a complaint is registered. In parallel, a questionnaire and the complaint is evaluated. If the questionnaire is returned within a fixed delay, it is processed ($process_questionnaire$), else it is discarded ($time_out$). Now based on the result of the evaluation, the complaint is processed or not. If processing is required, it is archived ($archive$). The task processing is delayed until the questionnaire is processed or a time-out has occurred. The task progress is checked during the task of $check_processing$. In the end, the task is archived. Note that WF-net allows sequential, conditional, parallel, and routing iterations. A large part of the literature on workflow nets has been devoted to checking soundness [VDA+11], which is a set of conditions to guarantee a clean termination of a process.

Let m, m' be two markings. We say that m' is greater than m, and write $m' \geq m$ iff for every place $p \in P$, $m'(p) \geq m(p)$. We denote by $[o]$ the marking such that $m(o) = 1$ and $m(p) = 0$ if $p \neq o$. Soundness is defined as follows.

Definition 7 (Workflow Nets-Soundness [Aal97; VDA+11])**.** *A workflow net is sound if and only if following requirements are satisfied*

— *Termination:* $\forall m, m_0 \xrightarrow{\delta} m : \exists m', m \xrightarrow{\delta} m' : m' \geq [o]$

— *Clean Termination:* $m_0 \xrightarrow{\delta} m \wedge m \geq [o] \implies m = [o]$

— *No dead transitions:* $\forall t \in T, \exists m : m_0 \xrightarrow{\delta} m$ *and* t *is enabled in* m

The first requirement states that from the initial state (a marking m_0 with a token in place i and all other places empty), it is always possible to reach a state with one token in place o. The second requirement states when an execution reaches a marking where o contains a token, all other places are empty. This is a way to guarantee that all parallel threads launched during the execution are terminated. The last requirement states that there are no dead transitions. The property guarantees the absence of deadlock in workflow nets. To summarize, soundness guarantees that, "every execution starting from an initial marking and reachable to a marking with k tokens on the initial place terminates properly, i.e. it can reach a marking with k tokens on final place, for an arbitrary natural number k" [VHSV04]. Generalized and structural soundness is decidable for WF-nets [VHSV04; ŢM05].

WF-nets extended with time [LS00] integrates time to workflows and allows to consider timed safety. A net is timely safe if a transition cannot produce a token in an already marked place. In the untimed setting, this is called contact freeness. The timed variant allows addressing questions of resource usage and performance. Timed safety shows that the timing constraints set on transitions prevent processes from competing for shared resources. Nets (in particular workflow nets) can rapidly become long and unreadable sequences of transitions depicting atomic tasks at a very low level. As already mentioned, the definition of complex tasks calls for the possibility to refine a task into a complex sub-task, i.e. another layer of orchestration. Some models of nets and workflows have addressed higher-order and hierarchy [Lom01; LS99]. This allows for hierarchical modeling to build a complex system by composing smaller sub-systems. Although these higher-order models provide mechanisms to orchestrate tasks, but rather than data, the process remains central to such systems.

Petri alone is already an interesting model. As mentioned earlier, they are the formal model used to give semantics to several business process notations. BPMN, for

instance, can be formally defined as a particular class of 1-bounded basic net, called fork-join net [Aal98]. However, Petri nets are not sufficiently expressive to model crowdsourcing systems, as they do not incorporate essential aspects such as data, time, and randomness.

2.3.2 Artifact Centric Approaches

Process centric based models focus on control flow, and most of the time does not consider data. To deal with this issue, several models centered on data have been proposed. Artifact centric models are data-centric models, i.e. focus is on the data, and how data is transformed by tasks. Artifact-centric business process models consider data as first-class citizens. The model considers addition, deletion, and manipulation of data by a set of tasks in the overall process. An artifact is a mechanism to record a key identifiable piece of information that is concrete, reliable, self-explanatory, and identifiable to be used and maintained to run the business. A collection of artifacts and services that modify this collection is called an artifact system. Very often, services are defined with rules that apply to one or several artifacts that meet given conditions. Contrarily to workflows, where the control flow of a process is explicitly represented by a flow relation, the control flow of a business artifact is based on rule realization and is hence declarative and implicit.

Business artifacts were originally developed by IBM [NC03] and proposed data-centric processing of artifact lifecycles. The lifecycle of all the artifacts in the business and their association describes the operational model of the entire business. An example of business artifact model, extracted from [BHS09] is shown in Figure 2.8 represented in the form Entity-Relationship (ER) diagram. ER diagram provides a framework to specify the models and to show the relationship among entities. The model is a fragment of process modeling of IT service providers that shows the specification of the vendor task and consists of the structure of the data, stakeholders, and conditions under which the tasks can be realized. Here, the *Vendor task* artifact has a set of attributes as schedule Id, planned start date, planned end date, status, etc. Vendor task artifact is also related to other sets of artifacts as *Vendor*, *Govt. approval*, etc. When an artifact is created, attribute values can be undefined or null values. As the execution progresses, the attribute value may be defined or overwritten. The artifacts are generally stored using relational or XML-based databases. The artifact model is stud-

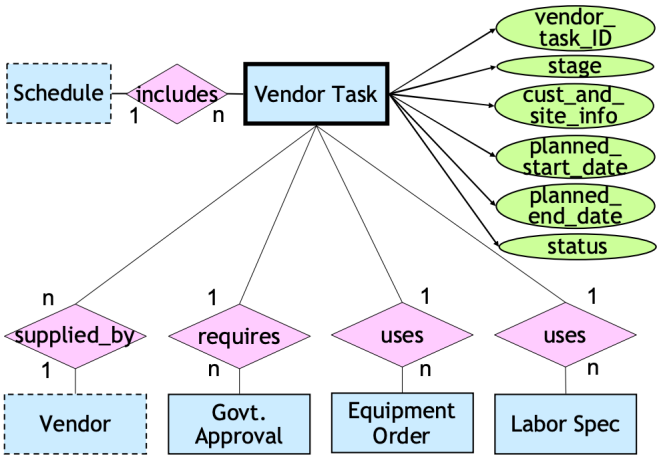


Figure 2.8 – Business artifacts represented as ER diagram specifying the vendor tasks, extracted from [BHS09].

ied extensively both from practical and theoretical perspectives. [Kum+03] proposes an adaptive document artifact system that formalizes collaborative work, in which a group of people collaborates to reach a business goal. The authors in [WK05] demonstrate the feasibility of a framework for document-driven workflow systems based on artifacts that require no explicit control flow and the execution of the process is driven by input documents. In a similar line, [GM05] proposes a formal model for document-based service to meet the business needs considering factors such as information that has to be exchanged, people, organizations, and the roles involved. [Hul+99] proposes an attribute centric workflow model called vortex workflow that allows the specifications of dataset manipulations and provides mechanisms to specify when services are applicable to a given artifact.

2.4 Data

Crowdsourcing provides a way to collect, retrieve, manipulate, and analyze data provided by human workers. Data is hence central to crowdsourcing systems. For example, a client provides an input dataset with a pre-defined objective. Data is manipulated and transformed by the workers and in the end, the final dataset is returned after completion of the task. In this section, we recall how datasets (also known as databases) are usually formalized and structured. From the beginning to the end of the

process, data is used as input, transformed, assembled, before being used as a final result. Data is not stored in any way. It is usually well structured and its transformation follows some rules. We also recall the basics of some formal tools used to query data and reason on the contents of datasets.

We use standard relational model [Cod72] to specify datasets. The relational model is simple, easy to understand yet possesses enough expressive power to store and manipulate data. The relational model considers *relations* as the data structure and comes with query capabilities, updates features, and supports integrity constraints. A database organized in terms of relations is known as a *relational database*. The relational model is a declarative model that abstracts control flows and does not consider compilation and optimization complexity needed to store and query data. The model is used to directly specify the information to be stored.

In the standard relational model [Cod72], data is organized in *datasets*, that follow *relational schemas*. In the context of the thesis, we assume finite set of domains $\mathbf{dom} = dom_1, \ldots, dom_s$, a finite set of attribute names $\mathbf{att}$ and a finite set of relation names $\mathbf{relnames}$. Each attribute $a_i \in \mathbf{att}$ is associated with a domain $dom(a_i) \in \mathbf{dom}$. A *relational schema* (or table) is a pair $rs = (rn, A)$, where rn is a relation name and $A \subseteq \mathbf{att}$ denotes a finite set of attributes. Intuitively, attributes in A are column names in a table, and rn the table name. The *arity* of rs is the size of its attributes set. A *record* of a relational schema $rs = (rn, A)$ is tuple $rn(v_1, \ldots v_{|A|})$ where $v_i \in dom(a_i)$ (it is a row of the table), and a *dataset* with relational schema rs is a multiset of records of rs. A *database schema* DB is a non-empty finite set of tables, and an instance over a database DB maps each table in DB to a dataset. Database schema specifies the structure of the database and the database instance specifies its actual content. We borrow an example from [AHV95] where a database schema **CINEMA** is defined as follows.

CINEMA = $\{Movies, Location, Pariscope\}$ where table $Movies$, $Location$, and $Pariscope$ have the following attributes:

attributes(Movies) = $\{Title, Director, Actor\}$

attributes(Location) = $\{Theatre, Address, Phone\ Number\}$

attributes(Pariscope) = $\{Theater, Title, Schedule\}$

Movies	*Title*	*Director*	*Actor*
	The Trouble with Harry	Hitchcock	Gwenn
	The Trouble with Harry	Hitchcock	Forsythe
	The Trouble with Harry	Hitchcock	MacLaine
	The Trouble with Harry	Hitchcock	Hitchcock
	...	...	...
	Cries and Whispers	Bergman	Andersson
	Cries and Whispers	Bergman	Sylwan
	Cries and Whispers	Bergman	Thulin
	Cries and Whispers	Bergman	Ullman

Location	*Theater*	*Address*	*Phone Number*
	Gaumont Opéra	31 bd. des Italiens	47 42 60 33
	Saint André des Arts	30 rue Saint André des Arts	43 26 48 18
	Le Champo	51 rue des Ecoles	43 54 51 60
	...	...	...
	Georges V	144 av. des Champs-Elysées	45 62 41 46
	Les 7 Montparnassiens	98 bd. du Montparnasse	43 20 32 20

Pariscope	*Theater*	*Title*	*Schedule*
	Gaumont Opéra	Cries and Whispers	20:30
	Saint André des Arts	The Trouble with Harry	20:15
	Georges V	Cries and Whispers	22:15
	...	...	...
	Les 7 Montparnassiens	Cries and Whispers	20:45

Table 2.1 – An example of **CINEMA** database, extracted from [AHV95].

The database is shown in Table 2.1. Each of the tables $Movies, Location$ and $Pariscope$ consists of rows where data is stored considering the domain of the attributes. Here, the $Movies$ table is of *arity* 3. The domain associated with each *attributes* of table $Movies$ is $dom(Title) \in strings, dom(Director) \in strings, dom(Actor) \in strings$. A *record* of table $Movies$ is tuple with values $The\ trouble\ with\ Harry,\ Hitchkok,\ Gwenn$ associated with attributes $Title, Director$ and $Actor$.

2.4.1 Reasoning on Data

First-order logic is a set of formal systems allowing reasoning on relations and hence on data. In the rest of the thesis, we will use sentences built with the following symbols :

— Variable. A Variable is a placeholder $v_1, v_2, u_1, u_2, ...$ to assign varying objects. Every possible value for a variable v_i belongs to a particular domain $dom(v_i)$.

— Logical Operators. Logical operator is a symbol used to connect two or more conditions. Logical operators are $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

— Logical quantifiers. A logical quantifier specifies the quantity of specimens having some property. The quantifiers are existential quantifier "$\exists$" (there exists), universal quantifier "$\forall$" (for all).
— Equality symbol. $\{=\}$ symbolizes the binary equality relation.
— Constant symbol. A constant symbol c denotes a fixed object whose value does not change over time.
— Function symbol. A function f denotes a $n - ary$ function that associates an object to a $n - tuple$ of objects.
— Relation and membership symbol. We denote by $rn(x_1, \ldots, x_k)$ a tuple from a relational schema rs, and write $rn(x_1, \ldots, x_n) \in D$ to denote the property that tuple $rn(x_1, \ldots, x_n)$ is a record of dataset D. Here, $\{\in\}$ denotes the membership symbol.
— Predicates. A predicate is a boolean function $P(x_1, \ldots, x_n)$ that evaluates to TRUE or FALSE under a particular interpretation of variables $x_1, \ldots, x_n$.

The formula $P(x_1, \ldots, x_k)$ means $x_1, \ldots, x_k$ satisfy predicate P. Note that the difference between a function and a predicate is that an application of function returns an object, while the application of predicate returns a TRUE or FALSE value. Signature of FO describes the non-logical symbols. The signature consists of a set of constant symbols, function symbols, relation symbols and predicate symbols.

A term in FO is a sentence of the form $t ::= v|c|f(t_1, \ldots, t_n)$ where,
— v is a variable name.
— c is a constant.
— $f()$ is a function, and $t_1 ... t_n$ are terms.

Then, FO formula F are defined as sentences of the form: $F ::= \top \mid \bot \mid F \vee F \mid \neg F \mid P(t_1, \ldots, t_n) \mid rn(x_1, \ldots, x_n) \in D \mid \forall v(F) \mid \exists v(F)$. Here, $t_1, \ldots, t_n$ are terms and $x_1, \ldots, x_n$ are variables. Formula may also contain the usual boolean connectors $\wedge, \rightarrow, \leftrightarrow$ with the usual meaning.

An *atomic formula* or *atom* is simply a predicate applied to a tuple of terms, i.e. an atomic formula is a formula of the form $P(t_1, \ldots, t_n)$ where P is a predicate, and $t_1, \ldots, t_n$ are terms. The formula $\exists x P(x)$ is *there exists* x, P and the formula $\forall x P(x)$ is *for all* x, P. Letting $X_1 = \{x_1, \ldots x_k\} \subseteq X$, we write $\forall \vec{X}_1$ instead of $\forall x_1 . \forall x_2 \ldots \forall x_k$. Similarly, we write $\exists \vec{X}_1$ instead of $\exists x_1 . \exists x_2 \ldots \exists x_k$. We follow this convention through out the thesis.

An *interpretation* is a function that attaches the meaning to the constant, function and relation symbols. Formally, an interpretation is defined as follows.

Definition 8 (Interpretation)**.**

- *An interpretation of a constant symbol c is an element of its domain dom.*
- *An interpretation of a function symbol f with arity n is a function that maps n elements of dom to another element of dom.*
- *An interpretation of a predicate symbol P with arity n is the set of n tuples of elements of dom for which dom is TRUE.*

Definition 9 (Assignment)**.** *An assignment in FO logic assigns values to variables. Each variable x in X has its own domain $dom(x)$. A variable assignment (for a fixed set of variables X) is a function μ that associates a value d_x from $dom(x)$ to each variable $x \in X$.*

Given a variable assignment μ, we say that ϕ hold under μ, and write $\mu \models \phi$ iff ϕ evaluates to TRUE under assignment μ. Given a FO formula, we illustrate below when the formula is TRUE or FALSE.

- An atomic (atom) $n - ary$ predicate $P(t_1, \ldots, t_n)$ predicate when applied to n terms, returns TRUE or FLASE. Predicates are used to describe the property of objects. For example, $P(x)$ can be a predicate that evaluates to TRUE if x is an even number.
- A universally quantified formula $\forall x : F$ asserts that a certain property holds for any value $v \in Dom(x)$ assigned to variable x. $\forall(x)P(x)$ indicates that P holds for every assignment of a value from $dom(x)$ to x.
- An existentially quantified formula $\exists x : F$ means that formula F holds for some assignment that assigns some value $v \in Dom(x)$ to variable x. $\exists(x)P(x)$ indicates P holds for some assignment of a value from $dom(x)$ to x.

Consider a FO formula $\varphi ::= \forall x(x^2 \neq 2)$. If the domain of x is real number, $dom(x) \in \mathbb{R}$, then the formula φ is FALSE because for an *assignment* μ such that $\mu(x) = \sqrt{2}$, we have $x^2 = 2$. If the domain of x is natural number $dom(x) \in \mathbb{N}$, then the formula φ evaluates to TRUE as an *assignment* $x = 2$ is not the square of any natural number.

Let predicate $C(x)$ holds if x is a crowd worker and $E(x)$ holds if x is an expert. Let D_{in} be an input dataset with relational schema $rn(x_1, \ldots, x_k)$ and D_{out} be an output dataset with relational schema $rn'(x_1, \ldots x_k, y_1, \ldots, y_n)$. We show some examples of FO logic with plain English interpretation as follows.

— All crowd workers are expert. $(\forall x)[C(x) \rightarrow E(x)]$
— Some crowd workers are expert. $(\exists x)[C(x) \wedge E(x)]$
— For all input record there exists an output record. $\forall x_1, \ldots, x_k \; rn(x_1, \ldots, x_k) \in D_{in} \rightarrow \exists y_1, \ldots y_n \; rn'(x_1, \ldots, x_k, y_1, \ldots, y_n) \in D_{out}$

Definition 10 (Free and bound variables)**.** *A variable occurrence is free in a formula if it is not quantified. Variables in the scope of the quantifiers are called bound variables.*

Variables in a FO logic formula may occur *bound* or *free*. For example, $\forall y P(x, y)$, the occurrence of variable x is free. On the other hand variable y is bound as it is universally quantified. A *quantifier free* formula is a formula that contains no quantifiers.

Generally, FO formulas are given in Prenex Normal Form (PNF). A formula is in PNF if it is written as a string of alternating *quantifiers* and *bound* variables called *prefix* of the formula and followed by a quantifier-free part called *matrix* of the formula. Formally, the Prenex Normal Form of FO is defined as follows.

Definition 11 (First-order in Prenex Normal Form)**.** *A* first-order formula *in Prenex Normal Form over a set of variables X is a formula of the form $\varphi ::= \alpha(X).\psi(X)$ where $\alpha(X)$ is an alternation of quantifiers and variable names in X, i.e. sentences of the form $\forall x_1 \exists x_2, ...$ called the* prefix *of φ and $\psi(X)$ is a quantifier free formula called the* matrix *of φ. $\psi(X)$ is a boolean combinations of* atoms *of the form $R_i(x_1, \ldots x_k)$, $P_j(x_1, \ldots x_n)$, where $R_i(x_1, \ldots x_k)$ are relational statements, and $P_j(x_1, \ldots x_n)$ are predicates, i.e. boolean function on $x_1, \ldots x_n$.*

For example, a formula $\varphi ::= \exists x_0 \exists x_1 \forall x_2 \; P_1(x_0, x_1, x_2) \wedge P_2(x_0, x_2)$ is in PNF. In this formula $\exists x_0 \exists x_1 \forall x_2$ is the *prefix* of φ and $P_1(x_0, x_1, x_2) \wedge P_2(x_0, x_2)$ is its *matrix*. Every formula in FO logic is equivalent to formula in a prenex normal form. For example, a FO formula in the form of $\exists x(\exists y P_1(x, y) \vee (\forall z \; P_2(z) \vee P_3(x)))$ is equivalent to a formula in a PNF form: $\exists x \exists y \forall z(P_1(x, y) \vee (\; P_2(z) \vee P_3(x))$.

Definition 12 (Satisfiability)**.** *A variable free formula is* satisfiable *iff it evaluates to true. A formula of the form $\exists x, \phi$ is* satisfiable *iff there exists a value $d_x \in dom(x)$ such that $\phi_{[x/d_x]}$ is satisfiable. A formula of the form $\forall x, \phi$ is satisfiable iff, for every value $d_x \in dom(x)$, $\phi_{[x/d_x]}$ is satisfiable.*

Definition 13 (Equisatisfiability)**.** *Two formulas ϕ and ψ are equisatisfiable iff ϕ is satisfiable if and only if ψ is satisfiable. Equisatisfiability of ϕ and ψ is denoted as $\phi \models\!\!\models \psi$.*

One can rewrite a FO formula into an equisatisfiable formula using miniscoping rules. These rules are syntactic transformations that apply regardless of any interpretation of the formula. They can be applied to move quantifiers as deep as possible inside a formula (this is interesting for skolemization [BW84], but we will not use it in this thesis) or conversely to transform a formula into an equivalent one in prenex normal from.

Definition 14 (Miniscoping [SVW16])**.** *Let φ, ψ, X be the first-order formulas, and assume that x does not occur freely in X. Then using miniscoping, quantifiers can be pushed in FO as follows.*

(i). $\exists x.(\varphi \vee \psi) \models\!\dashv (\exists x_1.\varphi) \vee (\exists x_2.\psi)$

(ii). $\exists x.(\varphi \ o \ X) \models\!\dashv (\exists x.\varphi) \ o \ X$ *with* $o \in \{\vee, \wedge\}$

(iii). $\forall x.(\varphi \wedge \psi) \models\!\dashv (\forall x_1.\varphi) \wedge (\forall x_2.\psi)$

(iv). $\forall x.(\varphi \ o \ X) \models\!\dashv (\forall x.\varphi) \ o \ X$ *with* $o \in \{\vee, \wedge\}$

For example, a formula in form $\exists x \forall y(P_1(x) \leftrightarrow P_2(y))$ can be transformed using miniscoping rules as $\exists x \ \neg P_1(x) \vee \forall y P_2(y)$

It is well known that the problem of satisfiability of a FO formula is undecidable in general [Chu+36; Tur37]. This can be proved by reduction from the halting problem for Turing Machine. A lot of effort has been put to find decidable fragments of FO logic. There exist several fragments of first-order logic for which satisfiability is decidable.

Monadic FO is a fragment of first-order logic in which all relation symbols in a formula take only one argument. All atoms in formulas are of the form $P(x)$. [Löw15] pioneer work shows that confinement to unary predicate symbols (Monadic FO) leads to decidability. However, the monadic FO fragment has very low expressive power. The *universal fragment* (resp. existential fragment) of FO is the set of formulas of the form $\forall \overrightarrow{X} \varphi$ (resp. $\exists \overrightarrow{Y} \varphi$) where φ is quantifier-free. We denote by $\forall$FO the universal fragment and by $\exists$FO the existential fragment. Checking satisfiability of the existential/universal fragment of FO is decidable and (co)-NP-complete when variables domain are restricted to real or discrete values [Bou+19]. One needs not restrict to existential or universal fragments of FO to get decidability of satisfiability. A well-known decidable fragment is FO^2, which uses only two variables [Mor75]. However, this fragment forbids atoms of arity greater than 2, which is a severe limitation when addressing properties of datasets. A recent extension of FO^2 called FO^2BD allows atoms of arbitrary arity, but

only formulas over sets of variables where at most two variables have an unbounded domain. Interestingly, FO^2BD formulas are closed under the computation of weakest preconditions for a set of simple SQL operations [Itz+17]. The *Bernays-Schonfinkel* (BS) fragment of FO is the set of formulas of the form $\exists\vec{Y}_1.\forall\vec{X}_2.\psi$, where ψ is quantifier-free, may contain predicates, but no equality. The *Bernays-Schonfinkel-Ramsey* fragment of FO [BS28] (BSR-FO for short) extends the BS fragment by allowing equalities in the matrix ψ. The satisfiability of a formula in the BS or BSR fragment of FO is NEXPTIME-complete (with respect to the size of the formula) [Lew80]. Algorithms to check satisfiability for a fragment of FO can be obtained by transforming formulas from that class into equivalent or equisatisfiable formulas of a decidable subclass. Two FO formulas ϕ and ψ are *equivalent* iff, for every variable assignment μ, $\mu \models \phi$ iff $\mu \models \psi$. Transformation of a formula from one class to another class is usually expensive and results in a blowup of the size of considered formulas.

Recent results [SVW16] exhibited a new fragment, called the *separated fragment* of FO, defined as follows: Let $Vars(A)$ be the set of variables appearing in an atom A. We say that two sets of variables $Y, Z \subseteq X$ are *separated* in a quantifier-free formula $\phi(X)$ iff for every atom At of $\phi(X)$, $Vars(At) \cap Y = \emptyset$ or $Vars(At) \cap Z = \emptyset$. A formula in the *Separated Fragment* of FO (SF-FO for short) is a formula of the form $\forall\vec{X}_1.\exists\vec{Y}_2.\ldots\forall\vec{X}_n.\exists\vec{Y}_n\phi$, where $\vec{X}_1 \cdots \cup \vec{X}_n$ and $\vec{Y}_1 \cdots \cup \vec{Y}_n$ are separated. The SF fragment is powerful and subsumes the *Monadic Fragment* and the BSR fragment. Every separated formula can be rewritten into an equivalent BSR formula (which yields decidability of satisfiability for SF formulas) but at the cost of an n-fold exponential blowup in the size of the original formula. Satisfiability of a separated formula φ is hence decidable [SVW16], but with a complexity in $O(2^{\downarrow n|\varphi|})$ where $2^{\downarrow n}$ is a tower $2^{2^{\cdots^n}}$ of exponential of size n.

2.4.2 Datalog

Datalog is another declarative programming language to reason on data that is syntactically based on Prolog. However, datalog is primarily used as a query language for deductive databases that make deductions based on rules and facts. It was invented to combine logic programming with relational database to construct formalism along with dealing with large databases. The most interesting feature of datalog is *recursion*. Datalog is defined as follows.

Definition 15 (Datalog [AHV95]). *A datalog rule is an expression of the form :*
$R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$
where $n \geq 1, R_1, \ldots, R_n$ *are relation names and* $u_1, \ldots u_n$ *are free tuples of appropriate arities. Each variable occurring in* u_1 *must occur in at least one of* $u_2, \ldots, u_n$.

A datalog program consists of a finite set of datalog rules. $R_1(u_1)$ is the head of the rule and $R_2(u_2), \ldots, R_n(u_n)$ forms the body. A datalog program defines the relations that occur in heads of rules based on other relations. The defined relations can also occur in the bodies of the rules and thus is recursive. Also to note that datalog programs may not use negation in the rule bodies. A simple example of datalog is as follows. We first define two facts.

— $parent(John, James)$
— $parent(James, Bill)$

The facts define, *John* is the parent of *james* and *james* is the parent of *bill*. We also define the below rules.

— $ancestor(X, Y) \leftarrow parent(X, Y)$
— $ancestor(X, Y) \leftarrow parent(X, Z), ancestor(Z, Y)$

The rule states the following.

— X is an ancestor of Y if X is a parent of Y.
— X is ancestor of Y if X is a parent of Z and Z is a ancestor of Y.

Now, on the above defined facts and rules, we fire the following query: $ancestor(John, X)$. It asks *who are the* X *that are the ancestor of* $John$*?* The query evaluation returns $James$ and $Bill$.

Query evaluation in datalog is based on first-order logic. However, datalog is not Turing complete and even has efficient algorithms to resolve queries [Ban+85; CSW95]. Yet some problems of datalog are undecidable. For instance, the boundedness problem, which asks whether a datalog program is equivalent to some non-recursive datalog program is undecidable [Hil+95]. Datalog is of great theoretical importance. Datalog provides a clean basis to model deductive systems and is used to study recursive queries. However, due to lack of negation, it is not adequate as practical query language [AHV95]. It cannot express even the first-order queries. Also, Datalog is not a programming language and very few applications have taken full advantage of the expressiveness of datalog [CGT+89]. However, a recent revival of interest has emerged, using datalog queries in various applications as data integration, declarative networking, information extraction, and program analysis [HGL11; Mei+20].

2.4.3 SQL

Structured Query Language (SQL) is the most practiced query language that provides the basis for the extension of the relational model. It was initially developed at IBM by Donald D. Chamberlin and Raymond F.Boyce on the foundations of the relational model by Edgar F.Codd [Cha12]. In 1986, SQL was first formalized by ANSI and the most recent version of the standard was published in 2019. It is widely used in mainframe and client-server relational database management systems. SQL is useful in handling structured data where data contains the relationship between entities and variables.

SQL is a query and data manipulation language. First, it allows us to define and create the structure of the database. Using SQL commands as $CREATE$, one can define the organization of the database and the tables. On each defined table, SQL also allows for data access control. Beyond tables definition and creation, the most interesting features of SQL is as a query language, that allows to *select* data from a table, or even produce new datasets obtained as *joins* on table contents. We give below some examples of SQL commands.

SELECT-FROM-WHERE

The basic building block of SQL queries is the SELECT-FROM-WHERE command. It selects the data from a table based on the filter condition (where clause). Consider a SQL query on the CINEMA database on table Movies (see Table 2.1).

```
SELECT Title
FROM Movies
WHERE Actor = 'Gwenn'
```

The query selects the *Title* from the *Movies* database where the actor name is *Gwenn* and returns answer as *"The Trouble with Harry"*.

JOIN

The most interesting command of SQL is JOIN that combines rows of two or more tables based on a common column between them. Consider a SQL query on the CINEMA database on table *Pariscope* and *Location*.

Address	*Title*	*Schedule*
31 bd.des Italiens	Cries and Whispers	20:30
30 rue Saint Andre des Arts	The Trouble with Harry	20:15
.	.	.
.	.	.

Table 2.2 – JOIN query result.

```
SELECT Location.Address, Pariscope.Title , Pariscope.Schedule
FROM Location
JOIN Pariscope
ON Location.Theater=Pariscope.Theater
```

The JOIN command returns all records with the common values on attribute $Theater$ from the tables $Location$ and $Pariscope$. The above query produces the Table 2.2. We list some of the SQL commands with descriptions as follows.

- CREATE DATABASE - Creates a new database.
- ALTER DATABASE - Modifies a database.
- CREATE TABLE - Creates a new table.
- ALTER TABLE - Modifies a table structure.
- DROP TABLE - Deletes a table.
- SELECT - Extracts data from a database.
- UPDATE - Updates data in a database.
- DELETE - Deletes data from a database.
- INSERT INTO - Inserts new data into database.
- CREATE INDEX - Creates a search key.
- DROP INDEX - Deletes an index.
- JOINS - Combines rows from two or more tables based on common column between them.

SQL also supports aggregation queries with commands such as MIN, MAX, COUNT, AVG, and SUM along with operators as AND, OR, and NOT. The MIN( ) clause returns the smallest value in a column and MAX( ) clause returns the largest value of the selected column. COUNT( ) returns the number of rows that matches a specific condition. The AVG( ) clause returns the average value of a column with numerical values and SUM( ) returns the sum value of a column value with numerical values. These operators can be added with AND, OR and NOT clauses.

COUNT

Consider a below example. The SQL query counts the movie title from the table $Movies$ where actor name is $Gwenn$.

```
SELECT COUNT(Title)
FROM Movies
WHERE Actor = 'Gwenn'
```

SQL works on these simple, yet powerful clauses that is easy to understand and implement. This makes SQL a more expressive language and is running on thousands of data-driven web applications. The crowdsourcing platforms are data driven and is composed of task which can be executed by machines using simple SQL queries. We will use the SQL queries as a building block to execute the tasks which can be automated in our model.

Remark. Edgar F. Codd first described the relational model for database management as an approach to manage data using a structure and language consistent with first-order logic [Cod02]. The query languages in the relational model are based on the theoretical foundations of relational algebra and relational calculus. Relational calculus is a formal query language similar to relational algebra. It is a declarative language that does not specify the order in which operations to be performed. Relational algebra uses algebraic structures based on logical rules for modeling data and defining queries on them. It defines operators to transform a set of input relations to an output relation. Relational algebra is a procedural language, in which order is specified to perform the operations. Relational algebra was created to implement the queries in an efficient way and is simple for machines to evaluate. The notable Codd's theorem *establishes that relational algebra and relational calculus have precisely equivalent expressive power* [Cod+72]. Relational algebra provides a basis for the relational database. The most popular query language SQL is based on the theoretical foundations of relational calculus. Relational calculus is essentially equivalent to first-order logic. Most of the database query languages in practice are based on the extensions of first-order logic. First-order logic has long been regarded as a fundamental tool to investigate and model the properties of data-based systems. In this thesis, we choose first-order logic to address properties on data and to theoretically investigate the properties of our crowdsourcing model.

2.4.4 Data Centric Models

Crowdsourcing systems are data-centric systems that collect, transform, and aggregate data. In such cases, termination of tasks with a proper output is of primary importance. This means that two things have to be considered. First, a data centric system *terminates* when forging its output data from answers collected by the crowd. Second that the produced data is *correct*, i.e. satisfies some criteria on expected legal output value or that confidence in the result is high enough. The correctness of data centric processes is widely studied and a comprehensive survey is presented in [CDGM13; DHV14]. We review below several models for which a notion of correctness has been formalized and studied.

Guarded Active XML [ASV09] (GAXML for short) is a specification paradigm where services are introduced in structured data. In such models, structured data embeds references to service calls. Services modify data when the guard is satisfied and replaces a part of the data with some computed value that may also contain references to service calls. Though GAXML does not address crowdsourcing nor task refinement, however, if services are seen as tasks, the replacement mechanism performed during calls can be seen as a form of task refinement. This model is very expressive, but restrictions on recursion allow for verification of Tree-Linear Temporal Logic (LTL). LTL [Pnu77] is an extension of propositional logic that defines the properties of a single execution of the system. An execution is a sequence of states $\rho = \{s_1, s_2, \ldots, s_n\}$. Each state carries some propositions, and LTL formulas define properties of the future at some step of a single execution of the system. LTL formula can express properties such as "*proposition p will be eventually true*", "*p will be true until q is true*", etc. It is a widely used tool in the formal verification of programs and systems. Tree LTL replaces propositions in LTL by properties of structured data seen as labeled trees. LTL is a fragment of first-order logic [Kam68]. LTL with first-order logic (LTL-FO) formulas is of form $\forall x_1, \ldots, x_k, \phi$ where ϕ is an LTL formula including FO statements. More precisely, LTL is equivalent to First-order Monadic Logic of Order (FOMLO) - a subset of FO where formulas are built using atomic propositions of the form $P(x)$, atomic relations between elements of the form $x_1 = x_2, x_1 < x_2$, Boolean connectives and first-order quantifiers $\exists x$ and $\forall x$. [Deu+06] consider verification of LTL-FO for systems composed of participants that communicate asynchronously over possibly lossy channels and can modify (append/remove records from local databases). Unsurprisingly, queues make LTL-FO undecidable, but bounding the queues allows for verification. Verification mechanisms

for LTL-FO were proposed for subclasses of artifacts with data dependencies and arithmetic in [DDV12; KV17].

The authors in [DDV12] propose a model called artifact systems. Artifact systems can simulate a counter machine so even the simplest properties are undecidable for this model. However, for a syntactic restriction called feedback-free artifact systems, verification of temporal properties is decidable. The properties can be expressed in LTL-FO and allow data dependencies and arithmetic that are essential in business processes. Feedback freedom prevents unbounded updates to a variable's current value that depends upon its history. It is designed to limit the data flow between occurrences of the same artifact variables at different times in runs of the system. The restriction results in the decidability of verification for LTL-FO properties of feedback-free systems. The worst-case hyperexponential complexity comes from the number of artifact variables. Generally, data centric systems need data transformation from one form to another. Data exchange is a process that takes the data structure in the source schema and transforms it into a data structure of the target schema based on a schema mapping. Schema mapping is a set of constraints or rules that state how data has to be transformed. Tuple Generating Dependency (TGD) is a widely used technique to define the constraints in relational database theory. Tuple generating dependency allows certain kind of constraint on relational database in the form of $\forall x_1, \ldots, x_n P(x_1, \ldots, x_n) \rightarrow \exists y_1, \ldots, y_m Q(x_1, \ldots, x_n, y_1, \ldots, y_m)$. Also, data transformations with arithmetic operations such as addition, multiplication, etc., called arithmetic schema mapping, are required by the data centric systems. Authors in [CKO13] study data exchange with arithmetic schema mapping and queries based on TGD. The author proposes a polynomial-time algorithm that tests whether, for a given source schema, there exists a target schema based on the transformation by the schema mapping. However, the constraints on data used by the authors are only existential and lack universal quantification on variables. Such restrictions in real crowdsourcing applications may limit addressing the properties on data. For example, in a crowdsourcing setting, we want to prove properties in the form: "every record in an output dataset returned by the crowd satisfies some property P". Here, the system requires universal quantification of variables in the output dataset. Indeed, business artifacts allow for data inputs during the lifetime of an artifact and describe legal relations on datasets before and after the execution of a task. However, it mainly considers static orchestrations of guarded tasks and does not consider higher-order constructs such as run time tasks

refinement. Further, LTL-FO verification focuses mainly on the dynamics of systems (termination, reachability), but does not address correctness.

[Bee+06] proposes a query language for querying business processes. The business processes are recursive functions that can call one another, and the verification that is performed is defined using a query language. This language BP-QL specifies shapes of workflow executions with a labeled graph and transitive relations among its nodes. A query Q is satisfied by a workflow W if there is an embedding of Q into the unfolding of W. While this model addresses the shape of executions, it is more focused on the operational semantics of workflows. Contrary, the verification scheme in a crowdsourcing setting is not centered on the shape of control flow, but rather centered on the contents of data manipulated by the crowd workers. It is more concerned by the denotational semantics of the specification, and on the question of termination. Relational data-centric dynamic systems are the systems where data is represented in the form of a relational database and the process is described in terms of atomic actions that evolve the database. In such systems, the execution of the atomic actions can invoke calls to external services to insert new data into the system. [Har+13] shows that such systems are undecidable in general and shows decidability in a restricted setting where the new data introduced is bounded along each run, although it needs not to be bounded in the overall system. More recently, [AV13] has proposed a model for collaborative workflows where participants have a local view of a global instance and collaborate via local updates. Overall, we can see that formal verification of a data centric system is decidable in restricted settings where the use of recursion or quantifiers is constrained.

2.4.5 Weakest Precondition

Another way to reason about correctness of a program in general, and hence also of a crowdsourcing system is to use systems of logical inference rules to deduce that initial conditions of all runs of a system to guarantee termination or correctness of results at the end of each execution. The seminal work for weakest preconditions is proposed by Edsger Dijkstra [Dij75]. The principle is to see statements of a language as a predicate transformer. Weakest precondition calculus allows for the construction of valid deductions of *Hoare* logic [Hoa69], i.e. prove the so-called Hoare triples.

Hoare logic provides formal system with a set of logical rules to reason about the correctness of computer programs. A *Hoare triple* is a statement of the form

```
{P}S{R}
```

Here, P and R are assertions and S is a command. P is called the *precondition* and R the *postcondition* and S is the *command*, code or some function. Precondition is a condition that needs to be true prior to the execution of some code. On the other hand, postcondition is a condition that should be true after the execution of some code. All the assertions are expressed in predicates. A Hoare triple $\{P\}S\{R\}$ states that when *precondition* P holds, then after execution of S, *postcondition* R holds. For expressive enough programming languages, proving a Hoare triple is undecidable.

One can interpret properties P and R as a specification of a program. In this context, we can say that program S is correct with respect to its specification iff it satisfies the assertions given by the pair (P, R). There are two ways to address correctness. **Partial Correctness** states that if an answer is returned, it will be correct. **Total correctness** says that returned answers are correct, but also adds the condition that program S terminates. Hoare triple provides a narrow notion of termination and using the standard notion of Hoare logic only partial correctness can be proved. To rephrase, Hoare triple states that if P holds before the execution of command S, then R will hold afterward or S does not terminate. The Weakest preconditions is a reformulation of Hoare logic that builds valid deductions of Hoare logic. It is defined as follows.

Definition 16 (Weakest Precondition)**.** *For a statement, S and a postcondition R, the weakest precondition is a predicate Q such that for any precondition P, $\{P\}S\{R\}$ if and only if $P \rightarrow Q$.*

The weakest precondition Q is the minimum condition on the state of a program (or in our case an input dataset) so that program S terminates and after execution of S, the *postcondition* is satisfied. In some sense, computing a weakest precondition is a way to synthesize a valid Hoare triple for a program S when the postcondition R is known. For a given sequence of instructions $S_1.S_2 \ldots S_k$ and a postcondition R that have to be met once $S_1.S_2 \ldots S_k$ is executed, one can inductively build Q_k as the weakest precondition for S_k and R, then Q_{k-1} as the weakest precondition for S_{k-1}, etc. Ending with a predicate Q_0, one then have to check if the initial state of the program $S_1.S_2 \ldots S_k$ meets requirements in Q_0.

Let us illustrate weakest precondition calculus on an example. Consider an instruction $x := x + 1$ and a postcondition $x > 0$. One valid precondition is $x > 0$ that is Hoare triple as $\{x > 0\}x := x + 1\{x > 0\}$ is satisfied. While the weakest precondition $wp(x := x + 1, x > 0)$ is $x > -1$. Here $wp = x > -1$ guarantees that after execution of function $x := x + 1$, the postcondition $x > 0$ will be true. By construction, a weakest precondition is unique. Finding a weakest precondition for a single instruction of a conditional statement can usually be automated. However, when programs involve while statements, computing weakest preconditions require for integration of loop invariant that usually have to be provided by humans. Hence, in general, weakest precondition calculus cannot be fully automated. We will see however that in the case of non-recursive specifications of crowdsourcing systems, this calculus can be automated. A recent extension of FO^2 called FO^2BD allows atoms of arbitrary arity, but only formulas over sets of variables where at most two variables have unbounded domain. Interestingly, FO^2BD formulas are closed under computation of weakest preconditions for a set of simple SQL operations [Itz+17]. We will use it to show that conditions needed for a non-terminating execution of our model are in $\forall FO$, and that $\forall$FO, $\exists$FO, BSR-FO, SF-FO are closed under precondition calculus.

2.5 Quality Assurance

Quality control in crowdsourcing is one of the primary concerns [Lea11]. For instance, consider a simple task: a requester asks to annotate a set of images in a predefined category on a crowdsourcing platform which acts as an input to train machine learning models. Errors in the annotated data can lead to errors (or quality problems) of the trained models. Henceforth, quality control is one of the main concerns in crowdsourcing systems. In general, quality control has to find a trade-off between two criteria, namely accuracy, and budget. Accuracy measures how precise or correct is the set of the answers returned by the crowd. Clients at a crowdsourcing platform usually have a limited budget to realize their tasks. A very high budget allows hiring a large pool of workers, but a limited budget forces them to use as many resources efficiently as possible. The question is hence how to obtain a reliable answer at a reasonable cost.

Difficulty in quality control comes from tasks and workers which correlate in complex ways to bring uncertainty to the system. First, tasks may be poorly designed, badly described, unclear, generic, or have specific requirements, subjective, difficult, etc. On

the other side, workers come with issues such as qualifications, skills set, experience, biases due to age, gender, sex, demographics, etc. One can also consider malicious workers, that even return the wrong answer on purpose [XFT15].

Due to these limitations, a single unknown worker cannot be trusted. To deal with the heterogeneity and uncertainty, tasks are usually replicated: each task is assigned to a *set of workers*. Redundancy is also essential to collect worker's opinions. These replicated work units are the basic elements of a larger task that can be seen as a poll. In this setting, one can safely consider that each worker executes his assigned task independently, and hence returns his own belief about the answer. As workers can disagree, the role of a platform is then to build a consensual final answer out of the values returned. Note that for each replication of a task, the accuracy of an answer is not measured at the level of a single worker, but actually depends on the collective contributions of all workers.

2.5.1 Aggregation Techniques

A technique that forges this final answer out of the worker's returned answers is called *aggregation technique*. We review some of them in the rest of the section. We first give the basic notations that we will use in the following sections.

General Setting

We consider a set of tasks $T = \{t_1, \ldots, t_n\}$ for which answers are unknown and need to be collected from a set of crowd workers $U = \{u_1, \ldots, u_k\}$. For each task t_j, the correct answer is a boolean value $a_j \in \{0, 1\}$. This answer is usually called the ground truth. For a task $t_j \in T$ the answers given by worker $u_i \in U$ is denoted by l_{ij}. We let y_j denote the *final answer* of a task t_j obtained by aggregating the answers of all workers. $L_j = \bigcup_{i \in 1..k} l_{ij}$ denotes the set of all answers returned by workers for task t_j, L denotes the set of all answers, $L = \bigcup_{j \in 1..n} L_j$. The goal is to forge the *final aggregated answer* $Y = \{y_j, 1 \leq j \leq n\}$ from the returned set of *answers* L by the crowd workers for all tasks.

Majority Voting

A natural way to derive a final answer from a set of answers is **Majority Voting** (MV), i.e. choose as a conclusion the most represented answer [EV11; Cao+12]. Given a task and a set of answers for that task, MV selects the final answer as the most voted answer among the provided solutions. The majority voting aggregation technique, to derive the final answer y_j for each task t_j is defined as follows.

Definition 17 (Majority Voting)**.**

$$\forall\, j \in n, \quad y_j = \underset{a \in \{0,1\}}{\operatorname{argmax}} \sum_{i=1}^{|L_j|} \mathcal{I}(l_{ij}, a) \tag{2.1}$$

where $\mathcal{I}$ is an indicator function, defined as follows

$$\mathcal{I}(x, y) = \begin{cases} 1 & if\ x = y \\ 0 & \textit{Otherwise} \end{cases} \tag{2.2}$$

The limitation of MV is that all answers have equal weight, regardless of the expertise of workers. If a crowd is composed of only a few experts, and of a large majority of novices, MV favors answers from several novice workers. However, in some domains, an expert worker may give a better answer than a novice and his answer should be given more weight. One can easily replace MV with a weighted vote. Let μ_{u_i} denotes the weight of the worker i. Then weighted majority voting is defined as follows.

$$\forall\, i \in n, \forall j \in n, \quad y_j = \underset{a \in \{0,1\}}{\operatorname{argmax}} \sum_{i=1}^{|L_j|} \mu_{u_i} \times \mathcal{I}(l_{ij}, a) \tag{2.3}$$

Higher weight is given to the more expert worker. However, assigning weight to workers raises the question of a measure for worker's expertise, especially when ground truth is not known.

Measuring Expertise: Golden Question

In general, expertise plays a major role to execute a task that may require some degree of specialization. For example, a Java programmer is required to code Java-based applications, a manager will be required to manage a project, an entomologist

is required to study insects, etc. A similar context applies to a crowdsourcing setting. Better knowledge about the expertise of the crowd workers can avoid uncertainty and improve the efficiency and accuracy of the answers. However, workers at crowdsourcing platforms are not necessarily known. Most of the crowdsourcing platforms do not have prior knowledge about the expertise and skills of their worker [Dan+18].

A way to obtain an initial measure of a worker's expertise is to use **Golden Questions** [Le+10; How06]. Golden questions are a set of tasks whose actual answers, known as *ground truth* are prior known. Several tasks with known *ground truth* are used explicitly or hidden to evaluate worker's expertise. Broadly, the evaluation mechanism is of two types, i.e. explicit and implicit. In the *explicit* approach, workers are asked to answer a set of test (golden) questions. The workers are then evaluated and only the qualified workers are allocated a particular "real" task. The qualification criteria is usually chosen by the client. In this setting, as qualifications tasks are unpaid, workers are often reluctant to realize these tasks. In the second *implicit* approach, a set of golden questions T_g (the tasks whose answer is known) is embedded in the actual set of tasks T_u (the tasks with unknown answers) to evaluate worker's expertise. The workers are evaluated and given weight depends on how they answered each task in T_g. The weighted aggregation method based on worker's expertise is then applied to derive the final answers for tasks T_u. As golden tasks T_g are hidden in standard crowdsourced tasks ($T = T_g \cup T_u$), they have to be paid as other tasks T_u. The advantage is that workers do not make a difference between qualification tasks and actual tasks. A drawback is that tasks T_g do not produce useful information as answers are already known but consume a part of the budget. Note that, *explicit* mechanism is not so appealing to workers and *implicit* way requires additional budget. Hence, both the mechanisms are not well suited and have a disadvantage on their own.

EM based techniques

Several papers have considered tools such as Expectation Maximization (EM) [DLR77] based techniques to aggregate answers. Given a statistical model that generates a set $\mathcal{X}$ of observed data, the EM algorithm aims at finding the unobserved data $\mathcal{Y}$ (missing data), and estimates hidden parameters θ. In the crowdsourcing context, the observed data $\mathcal{X}$ is the answers returned by the crowd, unobserved data $\mathcal{Y}$ is the final aggregated answers that need to be derived and θ is the hidden parameters such as expertise of the workers. EM is an iterative approach. Each round of the algorithm consists of two

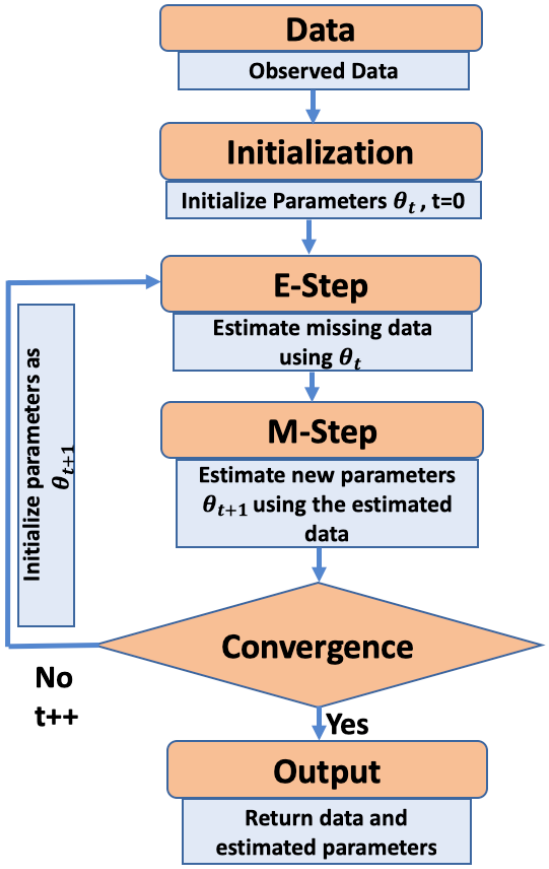


Figure 2.9 – Expectation Maximization algorithm general workflow.

phases: *Expectation step* and *Maximization Step*. The expectation step estimates the expected value of missing data given observed data and the current estimate of parameters. Then the maximization step aims to find the new value of the parameter based on the estimated missing data and the observed data. The two steps are repeated until convergence, i.e. the difference between the parameters at a step i and $i+1$ is smaller than a fixed threshold ϵ, i.e. $||\theta^{i+1} - \theta^i|| \leq \epsilon$. A coarse workflow for the EM algorithm is shown in Figure 2.9. We discuss in more detail the EM algorithm in Chapter 5.

A large part of the literature considers accuracy as a measure of workers expertise. We only highlight a few works addressing accuracy and a more complete survey can be found in [Zhe+17].

Accuracy: Accuracy is defined as the ratio of correctly answered tasks to the total number of tasks. Formally in our general setting (see Section 2.5.1), given n number of tasks, answer returned by a worker i for each task j be l_{ij} and the final answer evaluated by an aggregation algorithm (Majority voting, EM based technique) be y_j, we define the accuracy acc_i of a worker i as follows.

Definition 18 (Accuracy)**.**

$$acc_i = \frac{\sum_{j=1}^{n} ||l_{ij} = y_j||}{n} \tag{2.4}$$

Zencrowd [DDCM12] considers workers competences in terms of accuracy and aggregates answers using EM. The authors [Li+14; Ayd+14] model worker expertise as a single value between $[0, +\infty]$. The higher value implies the higher quality of workers. The author [Li+14] considers an optimization scheme based on Lagrange multipliers. Workers accuracy and ground truth are the hidden variables that must be discovered in order to minimize the deviations between workers answers and aggregated conclusion.

However, *accuracy* acc_i is not always a correct criterion to measure an expertise of a worker i. Consider a scenario where the task is to tag whether a mail is *spam* or not. Most of the mail in this scenario is *not spam*. In this case, even if a worker always tags a mail as *not spam*, his accuracy for the task will be very high. However, it is misleading. To cope with this drawback, some authors have proposed multi-dimensional evaluations, that consider ground truth in their measure. These measures are called *recall* and *specificity*.

The **recall** α_i of a worker i is the probability that worker i answers a task j as 1 when the ground truth is 1, i.e.

$$\alpha_i = Pr(l_{ij} = 1 | y_j = 1) \tag{2.5}$$

The **specificity** β_i of a worker i is the probability that worker i answers a task j as 0 when the ground truth is 0, i.e.

$$\beta_i = Pr(l_{ij} = 0 | y_j = 0) \tag{2.6}$$

Several works have shown that the quality of aggregated answers improves when expertize is measured in terms of recall and specificity rather than in terms of global accuracy. D&S [DS79] uses EM to synthesize answers that minimize error rates from a set of patient records. It considers recall and specificity to measure the worker's expertise and to aggregate the answers. The approach in [Ven+14] extends Bayesian

classifier combination (BCC) [KG12] with communities and is called CBCC. BCC is a general framework to combine the label in the context of the classification that explicitly models the relationship between each classifier model's output and the unknown truth label. In a crowdsourcing context, each model's output is replaced with the worker's answers and the idea is to use BCC to aggregate the answers. In the CBCC approach, each worker is supposed to belong to a particular (unknown) community and shares characteristics of this community (recall and specificity as a measure). The assumption helps to improve the accuracy of classification. [Ray+10] proposes a supervised learning approach when the ground truth is unknown. The work considers recall and specificity of workers and proposes a maximum-likelihood estimator that jointly learns a classifier, discovers the best experts, and estimates ground truth.

Task difficulty measures to which extent realizing a task requires an effort by a worker. For example, annotating a blurred image is more difficult than annotating a clear image. Most of the works cited above consider the expertise of workers but do not address task difficulty. An exception is GLAD (Generative model of Labels, Abilities, and Difficulties) [Whi+09] that proposes to estimate task difficulty as well as worker's accuracy to aggregate final answers. The authors recall that EM is an iterative process that stops only after convergence, but demonstrate that the approach needs only a few minutes to tag a database with 1 million images. The authors in [DLW13] consider the accuracy of workers and error in the execution of the tasks by the worker to aggregate the answers. Recall that considering accuracy as the worker's expertise can be misleading. Notice that in most of the works, task difficulty is not considered and expertise is solely modeled in terms of accuracy rather than using recall and specificity. In Chapter 5, we will propose EM-based aggregation mechanisms that consider recall and specificity of workers, and tasks difficulty as hidden parameters.

2.5.2 Budget optimization

The success of crowdsourcing platforms is based on the availability of human workers, available skills, and active participation and engagement of the crowd in the realization of the task. The human workers at platforms are demographically separated and it makes it hard for the administrators of the platform to physically track them. To motivate and push the participants, incentives are widely used. They can take the form of reputation systems, financial rewards, goodies, etc. [KSK16]. Financial rewards are the

most extrinsic motive which triggers the crowd workers most. The business platforms as AMT, Wirk, Figure Eight, etc. provide financial rewards as payment upon completion of a task. Requesters also provide bonuses if they are very happy with the performance of the crowd workers.

Financial compensation can take two forms: budget and bonus. A budget B_0 is an amount that is fixed apriori by a requester for the execution of all tasks and the bonus I_0 is an extra amount distributed by the requester to reward crowd workers after completion of a task. Note that, B_0 guarantees a minimum compensation for the realization of a task. On the other hand, the bonus amount varies as it is given after completion of a task by the requester and is totally based on the performance of the worker. If the system has an unlimited budget, it can hire an unlimited amount of workers. But constrained budget forces the system to use the budget at its best. In Chapter 6, we propose budget optimization schemes to make a trade-off between budget and accuracy.

As mentioned above, a single answer for a particular task is often not sufficient to obtain a reliable answer, and one has to rely on redundancy, i.e. distribute the same task to several workers and aggregate results to build a final answer. Standard *static* approaches on crowdsourcing platforms fix a prior number of k workers per task. Each task is published on the platform and waits for bids by k workers. There is no guideline to set the value for k, but two standard situations where k is fixed are frequently met. The first case is when a client has n tasks to realize with a total budget of B_0 units. Each task can be realized by $k = B_0/n$ workers. The second case is when an initial budget is not known, and the platform fixes an arbitrary redundancy level. In this case, the number of workers allocated to each task is usually between 3 and 10 [GM+16]. It is assumed that the distribution of workers is uniform, i.e. that each task is assigned the same number of workers. An obvious drawback of static allocation of workers is that all tasks benefit from the same work power, regardless of their complexity.

Generally, the database and machine learning communities focus more on data aggregation techniques and rather leave budget optimization apart. [KOS11] proposes an algorithm to assign tasks to workers, synthesize answers, and reduce the cost of crowdsourcing. The author considers a general model of crowdsourcing and provides an algorithm to decide task assignment to workers and also for inference of correct answers from the workers. It assumes that all tasks have the same difficulty and that worker's reliability is a consistent value in $[0, 1]$. The author provides an asymptotically

optimal graph construction in the form of a random regular bipartite graph and an optimal iterative method on the graph to allocate workers. CrowdBudget [TT+13] is an approach that divides a budget B among n existing tasks to achieve a low error rate and then uses MV to aggregate answers. Worker's answers follow an unknown Bernoulli distribution. The objective is to affect the most appropriate number of workers to each task in order to reduce the estimation error. The approach pre-allocates the workers to each task providing a bound on total estimation error based on budget. Static allocation needs not to allocate the same number of workers to each task. However, if the difficulty of a task is not known, it is difficult to choose a prior optimal allocation policy allowing a consensus for each task. Raykar et.al [RA14] introduce sequential crowdsourced labeling: instead of asking for all the labels in one shot, one decides at each step whether an evaluation of a task shall be stopped, and which worker should be hired. The model incorporates a Bayesian model for workers (workers are only characterized by their accuracy), and cost. In a similar line, [CLZ13] formalizes the problem as bayesian Markov Decision Process (MDP) to model workers reliability and incorporates it into budget allocation policy. In this MDP framework, set of states represent total budget consumed, and actions are the tasks that could be answered next. Transitions and their probability depend on the probability of answers returned by the workers, and the reward is defined in terms of the amount of incentive used to move to the next state (i.e. get more answers). The author [Li+16b] advocates that in some scenarios a requester might prefer getting fewer answers of the tasks with high quality rather than solving all the tasks. The proposed framework *Requallo* allows the requester to set specific requirements on the answers and then optimize the budget based on a one-step look ahead MDP framework. Note that computing a MPD out of a crowdsourcing system description results in a huge model, and that computing strategies to spend budget or hire workers on such model is not always a tractable approach.

Some works have considered deployment of tasks. CLAMSHELL [Haa+15] considers latency improvement. It affects workers to batches of tagging tasks and detects staggers. To speed up tasks completion, some batches are replicated. Pools are assembled and maintained by rewarding workers for waiting. This approach improves latency, but increases costs. [GP14] uses Markov decision processes to dynamically adapt a pricing policy so that batches of tasks are completed with the lowest latency within a fixed budget, or at the lowest price given some time constraint. [GIL16] proposes solution to compute the best static deployment policies in order to achieve an

optimal utility (i.e. a weighted sum of overall cost and accuracy) using sequencing or parallelization of tasks. Their exhaustive approach limits the number of workers and orchestrations that can be considered. [WRA20] is a recommendation technique for deployments, that allows parallelization of tasks, sequential composition, and use of machines to solve open tasks such as translation or text writing. Their approach considers fixed competence models, and provides static deployment solutions, building on optimization techniques. These deployments allow reduction of latency and improve quality of produced data.

Most of approaches studied consider cost optimization for batches of similar tasks such as image tagging, and do not consider cost optimization in situations where data is processed through a complex workflow. The author [Ber+15] proposes *Soylent*, a word processor which works in three phases: Find, Fix and Verify (FFV). It aids complex writing tasks by improving the quality of sentences. The workflow follows: In *Find* phase, workers detect positions of errors in the sentences, in *Fix* phase, workers rectify the mistakes, and in the end workers *verify* the mistakes. Cost is not considered in this setting. BudgetFix [TT+14] proposes to allocate budget across multi phases of a workflow. Given a budget of B, BudgetFix species the number of micro-task at each phase of a workflow along with a budget for each micro-task. The algorithm considers the Find-Fix-Verify workflow as a use case. However, it assumes prior knowledge of the difficulty of each of the tasks and considers workflow which works on one sentence at a time. Budgeteer [TT+15] is an extension of this work which attempts to manage cost across multiple workflows. The author considers processing multiple sentences using FFV workflow for each sentence. Note that, these approaches are very case-specific and do not provide a principled way to guarantee quality assurance in a general crowdsourcing workflow setting.

PART II

Data Centric Workflows for Crowdsourcing

CHAPTER 3

COMPLEX WORKFLOWS FOR CROWDSOURCING

Besides the simple human intelligence tasks such as image labeling, sentiment analysis, we foresee that crowdsourcing platforms have the capability to realize more intricate tasks. As explained in Chapter 2, the tasks on existing crowdsourcing platforms are simple and usually take a few minutes to an hour to complete. The next stage of crowdsourcing is to design more involved processes relying on the vast wisdom of the crowd. Indeed, many projects, and in particular scientific workflows, take the form of orchestrations of high-level composite tasks. Each high-level task can be seen individually as a data collection task, or as a processing of a large dataset, built as the union of results from independent easy micro-tasks. However, the coordination of these high-level tasks to achieve the final objective calls for more evolved processes. One can easily meet situations in which the result of a high-level task serves as an entry for the next stage of the overall process: for instance, one may want to remove from a picture dataset images of poor quality before asking workers to annotate them. Similarly, situations allow parallel processing of dataset followed by a merge of the obtained results. A typical example is the cross-validation of answers returned by different crowd workers. However, as noted by [Tra+15], composite tasks are not or poorly supported by existing crowdsourcing platforms. Crowdsourcing markets such as Amazon Mechanical Turk [1] (AMT), Foule Factory [2], CrowdFlower [3], etc. already propose interfaces to access crowd, but the formal design and specification of crowd-based complex processes are still in their infancy.

Many projects cannot be described as collections of repetitive independent micro-tasks: they require specific skills and collaboration among participants. We call such

1. https://www.mturk.com
2. https://www.foulefactory.com
3. https://www.crowdflower.com

projects “complex tasks”. The typical shape of complex tasks is an orchestration of high-level phases. For example - tag a database, then find relevant records, and finally write a synthesis. Each of these phases requires specific skills, can be seen at its level as a new objective on its own, and can be decomposed into finer choreographies, up to the level of assembly of micro-tasks. The workflow of such processes is hence dynamic and shall consider worker skills, availability of workers, and the produced output data, but also their knowledge about processes themselves. The first challenge is to fill the gap between a high-level process that a requester wants to realize and its implementation in terms of micro-tasks composition. Moving from one description level to the other is not easy, and we advocate the use of the expertise of the crowd for such refinement. This can be achieved with higher-order answers, allowing a knowledgeable worker to return an orchestration of simpler tasks instead of a crisp answer to a question.

In this chapter, we formalize the orchestration of complex tasks into smaller sub-tasks and define a model for crowd-based projects called “complex workflows”. Here is the outline of the chapter:

— We start with two higher-order examples of a complex task and its orchestration into smaller sub-tasks in Section 3.1.
— We describe the basic preliminaries to lay out the foundation of the chapter in Section 3.2. The notations are used throughout the thesis.
— We formalize the notion of workflow and the rewriting of tasks in Section 3.3.
— In Section 3.4, we define the operational semantics of complex workflows and we conclude in Section 3.5.

3.1 Higher Order Example

Our objective is to provide tools to develop applications in which human actors are involved to resolve tasks or propose solutions to complete a complex task. The envisioned scenario is the following: a client provides a coarse grain workflow depicting important phases of a complex task to process data and a description of the expected output. The tasks can be completed in several ways, but cannot be fully automated. It is up to a pool of crowd workers to complete them or to refine the tasks up to the fine-grain level where high-level tasks are expressed as orchestrations of basic simple tasks. To illustrate the needs for complex workflows, refinement, and human interactions, we provide two examples. We start with a simple example implementing an actor popularity pool and then we provide a real field example - the SPIPOLL case that uses higher-order.

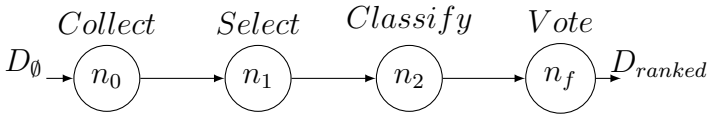


Figure 3.1 – A simple actor popularity poll.

3.1.1 A simple example: the actor popularity poll

A client (for instance a newspaper) wants to rank the most popular actors of the moment, in the categories comedy, drama, and action movies. The ranking demand is sent to a crowdsourcing platform as a high-level process decomposed into three sequential phases: first a collection of the most popular actors, then a selection of the 50 most cited names, followed by a classification of these actors in comedy/drama/action category. The ranking ends with a vote for each category, that asks contributors to associate a score to each name. The client does not input data to the system, but has some requirements on the output: the output is an instance of a relational schema $R = (name, cites, category, score)$, where $name$ is a key, $cites$ is an integer that gives the number of $cites$ of an actor, $category$ ranges over $\{drama, comedy, action\}$ and $score$ is a rational number between 0 and 10. Further, for an output to be consistent, every actor appearing in the final database should have a $score$ and the number of $cites$ greater than 0. From this example, one can notice that there are several ways to collect actor's names, several ways to associate a category tag, to vote, etc. However, the client's needs are defined in terms of high-level tasks, without information on how the crowd will be used to fulfill the demand. This simple sequential task orchestration is depicted in Figure 3.1. The workflow starts from an empty input dataset $D_\emptyset$, and should return the desired actor popularity data in a dataset D_{ranked}.

The model proposed in Section 3.4 is tailored to realize this type of application with the help of workers registered on a platform. Workers can either provide their answers on how to decompose difficult tasks or input their knowledge and opinion on the platform.

3.1.2 A real field example: the SPIPOLL initiative

We provide a real field example, where managing task decomposition, orchestration, and workers contributions are key issues. We study the SPIPOLL initiative[4], a participatory science project. The project aims at collecting quantitative data on pollination by flowering insects to measure the diversity and network structure of pollination in France. This task is not trivial for several reasons. First of all, collecting information on insects requires a huge work power to collect significant samples throughout the whole country. Second, the project lacks work power to sort, classify, huge datasets, and then derive conclusions from the observed populations. While volunteers can help to sort pictures of insects, some rare taxons can only be recognized by experts. The objective of SPIPOLL is to organize collaborative work involving ordinary people, trained volunteers, and experts to build significant datasets and extract information on pollinating insect populations. We believe that the project can benefit from advances in data-centric crowdsourcing platforms, in particular for the design and automation of complex data acquisition procedures.

The high-level description of SPIPOLL tasks is: acquire data (pictures), classify them, and publish the results. The result of classification should be a set of images with location and time tags, together with a taxon chosen among a finite set of species. The data acquisition and classification protocols specified by SPIPOLL fit well with the core idea of complex tasks. The protocol proposed by SPIPOLL is the following. Volunteers stand for a certain duration (usually 20 minutes) in a place frequented by insects (bushes, flower beds,...) and take pictures of insects pollinating on a flower at that place. Once the phase is completed, the observer uploads his pictures on a server. This first phase results in huge collections of pictures, but not all of them are exploitable. The obtained result after the first phase is a dataset which records are of the form of $R = \{image, place, time\}$. Data collected from various sources are used as inputs for a second phase. The second phase's objective is to classify pictures according to their quality, i.e. identify images that will help to find a correct identification for the represented insect. As the images are collected from diverse and unknown observers, there is a need to rank and tag the respective image based on their quality, i.e. associate with each image a tag $quality = \{poor, average, good, best\}$. Generally, for these tasks, considering the bias among the workers, a typical image is given to

4. http://www.spipoll.org/

k different workers. The workers give their opinion and tag each of the obtained images. The third phase aggregates the result obtained in the second phase and gives a final verdict for each picture. The *majority voting* technique is widely used to reach a common consensus among the workers. The fourth phase removes images with $poor$ quality from the dataset. The fifth phase categorizes the insect images with sufficient quality obtained as input from the fourth phase into different species categories, $category = \{category_0, \ldots, category_l\}$. Similar to the second phase, an image is distributed to k unique workers who are asked to tag a $category$ to the insect image. The output of this phase serves as input for a sixth phase that aggregates the result by majority voting to obtain the $final\ category$ for each picture and then passes the result for the final assessment. The final assessment for each image is performed by the experts or scientists for validation and then the obtained results are published. These types of complex tasks can be often found when there is a need for data collection, data cleaning, and then processing the data based on human intelligence.

The SPIPOLL example raises several observations on the realization of complex tasks in a crowdsourcing environment. First, one can note that some of the tasks need to be completed sequentially, and on the other hand some tasks can be executed in parallel. For example, in the $Spipoll$ case study, images can only be annotated after they have been collected by observers. As a result, the first phase and the second phase can only be executed sequentially. However, images can be annotated in parallel (second, fifth phase) by different sets of workers. Another remark is that some recurrent orchestration patterns appear, such as dataset distribution, tagging, and aggregation. These work distribution patterns are typical examples where crowdsourcing platforms are particularly adapted to improve quality and delays in data analysis. The dataset obtained from the observers can be large and annotating such big datasets is usually a task that cannot be performed by a single worker. Most often, volunteer contributors in crowdsourcing platforms spend little amounts of time on the platform. Hence the annotation phase needs to be decomposed, for instance by splitting large datasets into several chunks of reasonable size. These small chunks of data are then distributed and tagged in parallel by several workers, before aggregation. In this setting, refinement of heavy tasks into several smaller concurrent easy tasks saves a lot of time.

Another lesson learned from the SPIPOLL case study is that some tasks may require special skills, such as expertise on insects to be completed. The validation task in the above example requires specialized people to judge the validity of the proposed

taxons. Hence, the system should consider the expertise of the worker when allocating tasks. Another thing to note is that the example depicted here is data-centric, i.e. the role of each high-level or low-level task is to manipulate and transform data (select, tag,...). One can also observe that all tasks do not require human intelligence, and some of them can be easily performed by machines. For example, consensus between the workers is performed by applying majority voting techniques. Other tasks such as the selection of best pictures are like SQL queries and can be automated. Thus, the system is a mixture of human and machine-powered intelligence. Generally, machine-powered tasks are deterministic in nature, and tasks devoted to humans come with uncertainty.

As mentioned above, the output of one task acts as input to another task, which leads to causal dependencies among tasks. Besides, the execution of one task may affect the output of another task. An erroneous output of one task can jeopardize the execution of several successive tasks and in the long run, may also halt the whole process. Hence, this calls for the use of verification techniques to ensure consistency and guaranteed output for each of the task execution.

3.2 Preliminaries

A *complex workflow* is defined as an orchestration of *tasks*, specified by a *client* to process *input* data and return an *output* dataset. A client is a person, requester, or an organization who wants a crowdsourcing platform to realize a service defined as a complex task. We assume a fixed and finite pool $\mathcal{U}$ of workers, and a prior finite list of competencies $\mathbf{comp}$. A worker $u \in \mathcal{U}$ completes or refines some tasks according to its *skills*. We hence define a map $sk : \mathcal{U} \rightarrow \mathbf{comp}$. Notice that $sk(u)$ is a set, i.e. a particular competence $c \in sk(u)$ needs not be exclusive. We adopt this simplistic model of worker's competencies for the clarity of the model, but more evolved representations of skills exist and could be easily integrated into the model. For instance, [MGM16] proposes a hierarchy of competencies to reflect a natural ranking of the expertise. However, this thesis does not consider skills classification nor management of competencies and just views the skills of a worker as a set of keywords.

A task t is a work unit designed to transform input data into output data. A task can be of several types: it can be a high-level description of a stage in a complex task, a very basic atomic task that can be easily accomplished by a single worker (for example,

tagging images), a task that can be fully automated, or a complex task that requires an orchestration of sub-tasks to reach its objective. Tasks at crowdsourcing platforms can be accomplished by either humans, i.e. *workers* if they require human intelligence, or by machines, i.e. the tasks that can be automated.

We define a set of tasks $\mathcal{T} = \mathcal{T}_{ac} \uplus \mathcal{T}_{cx} \uplus \mathcal{T}_{aut}$ where $\mathcal{T}_{ac}$ is a set of *atomic tasks* that can be completed in a single step by a worker, $\mathcal{T}_{cx}$ is a set of *complex tasks* which need to be decomposed into an orchestration of smaller subtasks to produce an output, and $\mathcal{T}_{aut}$ is a set of *automated tasks* that are performed by a machine (for instance some database operation (selection, aggregation, union, projection, etc.)). $\mathcal{T}_{aut}$ do not require a contribution of a worker to produce output data from input data, and tasks in $\mathcal{T}_{ac}$ and $\mathcal{T}_{aut}$ cannot be refined. We impose constraints on skills required to execute a task with a map $T_{cs} : \mathcal{T} \rightarrow 2^{\mathbf{comp}}$, depicting the fact that a worker u is allowed to realize or decompose task t if it has the required competences, i.e., if $T_{cs}(t) \subseteq sk(u)$. One can, however, consider that every worker has all competences, and can perform any task within the system. Within this setting, one needs not define worker's competencies, nor attach skills constraints to tasks. However, for practical use of a crowdsourcing platform, one usually wants to obtain the best possible results, which calls for clever management of skills, incentives, etc. Now we define the task refinement. Let $t \in \mathcal{T}_{cx}$ be a complex task, and let $C_t = T_{cs}(t)$ be the set of competencies that allows a worker to realize a task t. We advocate that crowdsourcing platforms should allow higher-order answers.

For instance, in the Spipoll case (see Section 3.1), some intricate tagging tasks may require some special interactions among workers, followed by an expert's validation. Such tasks can be orchestrated in a workflow of bounded size assembling nodes which labels belong to a finite alphabet of tasks. The possibility is already in use in the world of crypto trading platforms such as Kryll that allows users to define simple trading bots using a block diagram language [Kry18]. We assume that a competent worker u knows a set of finite orchestrations depicting appropriate refinements of a task t. We denote by $Profile(t, u)$, this set of finite workflows. Note that a profile in $Profile(t, u)$ needs not to be a workflow of large size, and may even contain workflow with a single node. In such cases, the refinement simply replaces $t \in \mathcal{T}_{cx}$ with a single atomic tagging task $t' \in \mathcal{T}_{ac}$, meaning that u thinks that the task is easy, and wants it to be realized by another worker with specific skills.

We illustrate the refinement with an example. Assume a task $t \in \mathcal{T}_{cx}$ with an objective is to tag a (huge) dataset D_{in}. Here, $Profile(t, u)$ contains a workflow that first decomposes D_{in} into K small tables, then inputs these tables to K tagging tasks in $\mathcal{T}_{ac}$ that can be performed by humans in a reasonable amount of time, and finally aggregates the K obtained results. This refinement has many practical applications, for instance consider the following scenario: a complex task t_{like} asks to rank large collections of images of different animals with a score between 0 and 10. The relational schema for the dataset D used as input for t_{like} is a collection of records of the form $Picdata(nb, name, kind)$ where nb is a key, $name$ is an identifier for a picture, $kind$ denotes the species type obtained from the former annotation of data by crowd workers. A worker u decides to divide dataset D into three disjoint datasets that consist of pictures of cats, pictures of dogs, and pictures of other animals respectively. The three datasets are independent and can be ranked separately by the workers and in the end, results are aggregated. Figure 3.2 represents a possible profile to refine task t_{like}. A task t_{like} is rewritten in a workflow with four nodes. Node n_0 is an automated task that splits the original dataset into a dataset containing pictures of dogs, cats, and other animals. Nodes n_1, n_2, n_3 are occurrences of tagging tasks for the respective animal kinds, and node n_f is an automated task that aggregates the results obtained after realization of preceding tasks.

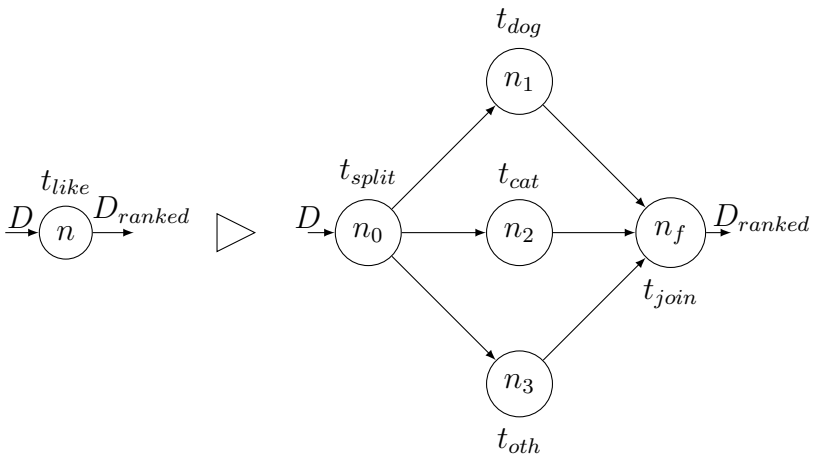


Figure 3.2 – A profile for refinement of task t_{like}.

3.3 Workflow Formalization

In this section, we formalize the notion of *workflow*, and *complex workflow*. This model is inspired by artifacts systems [DDV12], but uses higher-order constructs (task decomposition), and relies on human actors (the *crowd workers*) to complete tasks. We assume a *client* willing to use the power of crowdsourcing to realize a *complex task* that needs human contribution to collect, annotate, or organize data. This complex task is specified as a workflow that orchestrates elementary tasks or other complex tasks. The client can input data to the system (i.e. enter a dataset D_{in}) and have prior knowledge on the relation between the contents of the input and the plausible outputs returned after completion of the complex workflow. This scenario fits several types of applications such as opinion polls, citizen science participation, etc. High-level answers of workers are seen as workflow refinements, and elementary task realizations are simple operations that transform data. During the execution of a complex workflow, we consider that each worker is engaged in the execution of at most one task.

Definition 19 (Workflow)**.** *A* workflow *is a labeled acyclic graph* $W = (N, \rightarrow, \lambda)$ *where* N *is a finite set of* nodes*, modeling occurrences of tasks,* $\rightarrow \subseteq N \times N$ *is a precedence relation, and* $\lambda : N \rightarrow \mathcal{T}$ *associates a task name to each node. A node* $n \in N$ *is a* source *iff it has no predecessor and a* sink *iff it has no successor.*

We fix a finite set of tasks $\mathcal{T}$, and denote by $\mathcal{W}$ the set of all possible workflows over $\mathcal{T}$. Intuitively, if $(n_1, n_2) \in \longrightarrow$, then an occurrence of task named $\lambda(n_1)$ represented by n_1 must be completed before an occurrence of task named $\lambda(n_2)$ represented by n_2, and that data computed by n_1 is used as input for n_2. We denote $min(W)$ the set of sources of W, by $succ(n_i)$ the set of successors of a node n_i, and by $pred(n_i)$ its predecessors. The *size* of W is the number of nodes in N and is denoted $|W|$. We assume that when a task in a workflow has several predecessors, its role is to aggregate data provided by preceding tasks, and when a task has several successors, its role is to distribute excerpts from its input dataset to its successors. With this convention, one can model situations where a large database is to be split into smaller datasets of reasonable sizes that are then processed independently. We denote by $W \setminus \{n_i\}$ the restriction of W to $N \setminus \{n_i\}$, i.e. a workflow W from which node n_i is removed along with all edges which origins or goals are node n_i.

We assume some well-formedness properties of workflows:

1. Every workflow has a single *source* node n_{int}.
2. Every workflow has a single *sink* node n_f. Informally, we can think of n_f as the task that returns the dataset computed during the execution of the workflow.
3. There exists a path from every node n_i of W to the *sink* node n_f. The property prevents from launching tasks which results are never used to build an answer to a client.

We define a *higher-order* answer as a *refinement* of a node n in a workflow by another workflow. Intuitively, n is simply replaced by W' in W.

Definition 20 (Refinement)**.** *Let $W = (N, \longrightarrow, \lambda)$ be a workflow, $W' = (N', \longrightarrow', \lambda')$ be a workflow with a unique source node $n'_{src} = min(W')$ and a unique sink node n'_f such that $N \cap N' = \emptyset$. The refinement of $n \in N$ by W' in W is the workflow $W_{[n/W']} = (N_{[n/W']}, \longrightarrow_{[n/W']}, \lambda_{[n/W']})$, where*

— $N_{[n/W']} = (N \setminus \{n\}) \cup N'$

— $\lambda_{[n/W']}(n_i) = \lambda(n)$ *if* $n_i \in N, \lambda'(n_i)$ *otherwise*

— $\rightarrow_{[n/W']} = \rightarrow' \cup \{(n_1, n_2) \in \rightarrow \mid n_1 \neq n \wedge n_2 \neq n\} \cup \{(n_1, n'_{src}) \mid (n_1, n) \in \rightarrow\} \cup \{(n'_f, n_2) \mid (n, n_2) \in \rightarrow\}$

To illustrate the notion of refinement, consider the example of Figure 3.3. In the workflow at the left of the figure, node n_1 is replaced by the profile of Figure 3.2 for task t_{like}. The result is the workflow on the right of Figure 3.3.

Based on the definition of workflow and refinement, we now define the complex workflow as follows.

Definition 21. *A* Complex Workflow *is a tuple $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ where $\mathcal{T}$ is a set of tasks, $\mathcal{U}$ a finite set of workers, $\mathcal{R} \subseteq \mathcal{T} \times 2^{\mathcal{W}}$ is a set of rewriting rules, and $sk \subseteq (\mathcal{U} \times \mathcal{R}) \cup (\mathcal{U} \times \mathcal{T}_{ac})$ defines workers competences. W_0 is an initial workflow, that contains a single* source *node n_i and a single* sink *node n_f.*

We assume that in every rule $(t, W) \in \mathcal{R}$, the labeling λ of W is injective. This results in no loss of generality, as one can create copies of a task for each node in W, but simplifies proofs and notations afterward. Further, W has a unique source node $src(W)$. The relation sk specifies which workers have the right to perform or refine a particular task. This encodes a very simple competence model. The thesis does

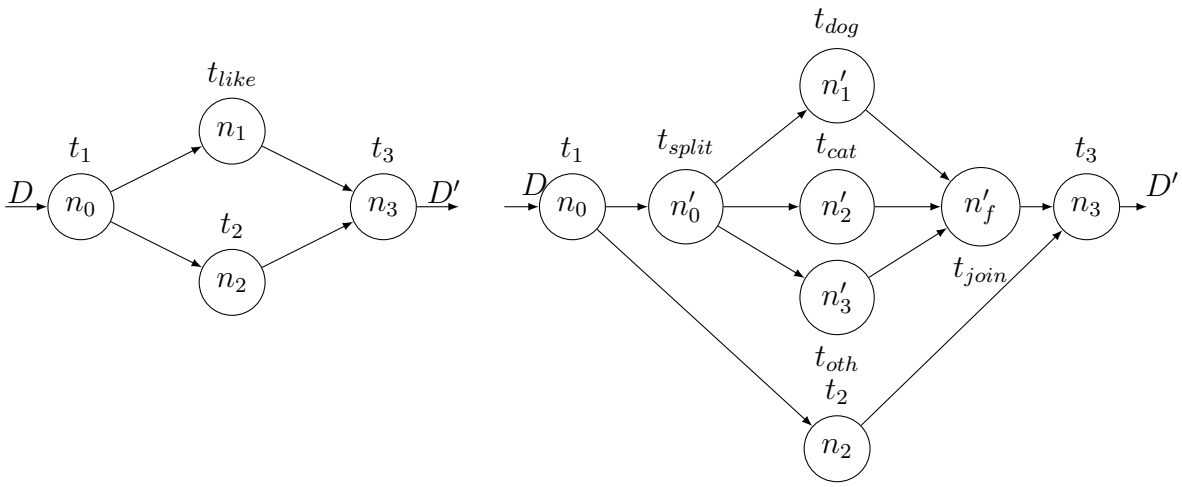


Figure 3.3 – A refinement of node n_1, replaced by the profile for task t_{like} in Figure 3.2.

not focus on the skill modeling of workers at crowdsourcing platforms and we refer to [MGM16] for further studies on competencies and more elaborated competence models.

Let us consider the example of Figure 3.3-left: a workflow contains a complex task t_{like} whose objective is to rank large collections of images of different animals with a score between 0 and 10. The relational schema for the dataset D used as input for t_{like} is a collection of records of the form $Picdata(nb, name, kind)$ where nb is a key, $name$ is an identifier for a picture, $kind$ denotes the species obtained from the former annotation of data by crowd workers. Let us assume that a worker u knows how to handle task t_{like} (i.e. $(u, t_{like}) \in sk$), and wants to divide dataset D into three disjoint datasets containing pictures of cats, dogs, and other animals, rank them separately, and aggregate the results. This is captured by the rule $R = (t_{like}, W_{like})$ of Figure 3.1-b, where node n_0 is an occurrence of automated tasks that splits an input dataset into datasets containing pictures of dogs, cats, and other animals, n'_1, n'_2, n'_3 represent tagging tasks for the respective animal kinds, and node n'_f is an automated task that aggregates the results obtained after realization of preceding tasks. A higher-order answer of worker u is defined as an application of rule R to refine node n_1 in the original workflow with W_{like}. The result is shown in Figure 3.3-right.

3.4 Operational Semantics

In this section, we define the operational semantics for complex workflows. A complex workflow considers the set of tasks, task constraints, worker skills, and comes with a set of operational rules. At each node of the workflow, data is transformed using data operations. Operational rules act as a guiding principle in the execution of a complex workflow. The semantics is defined with four rules - worker assignment, atomic task completion, automated task completion, and refinement. We also define configurations which represent the states of the complex workflow. An execution of an operational rule simply updates the current configurations of the workflow.

3.4.1 Data operations

We introduced the standard data representation using relational schema in Chapter 2, i.e. a representation of records by tuples of the form $rn(a_1, \ldots, a_n)$, where rn is a relation name and attributes $a_1, \ldots, a_n$ that fulfill the constraints of the legal domain of relation rn. The restriction by the client puts restrictions to reduce the range of legal input data or on the expected output of a complex workflow. Termination of a complex workflow depends upon the properties of a dataset and the transformation of data contents during configuration changes.

Here we first detail how automated and simple tasks are realized and process the data input to that task to produce output data. At every node n representing a task $t = \lambda(n)$, the relational schema of all input (resp. output) datasets are known and denoted $rs_1^{in}, \ldots, rs_k^{in}$ (resp. $rs_1^{out}, \ldots, rs_k^{out}$). We denote by $\mathcal{D}^{in} = D_1^{in}, \ldots, D_k^{in}$ the set of datasets provided by predecessors of t as an input to task t, and by $\mathcal{D}^{out} = D_1^{out}, \ldots, D_q^{out}$ the set of output datasets computed by task t. During a run of a complex workflow, we allow tasks executions only for nodes which inputs are not empty. The contents of every D_i^{out} produced during the execution of a task is the result of one of the operations below:

SQL-LIKE OPERATIONS : We allow standard SQL operations:

— **Selection:** For a given input dataset D_i^{in} with schema $rn(x_1, \ldots, x_n)$ and a predicate $P(x_1, \ldots, x_n)$, compute $D_j^{out} = \{rn(x_1, \ldots x_n) \mid rn(x_1, \ldots, x_n) \in D_i^{in} \wedge P(x_1, \ldots, x_n)\}$.

— **Projection:** For a given input dataset D_i^{in} with schema $rn(x_1, \ldots, x_n)$ and an index $k \in 1, \ldots, n$ compute $D_j^{out} = \{rn(x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n) \mid rn(x_1, \ldots, x_n) \in D_i^{in}\}$.

— **Insertion/Deletion:** For a given input dataset D_i^{in} and a fixed tuple $rn(a_1, \ldots, a_n)$, compute $D_j^{out} = D_i^{in} \cup \{rn(a_1, \ldots, a_n)\}$ (resp. $D_j^{out} = D_i^{in} \setminus \{rn(a_1, \ldots, a_n)\}$)
— **Union:** For two input dataset D_i^{in}, D_k^{in} with schema $rn(x_1, \ldots, x_n)$, compute $D_j^{out} = D_i^{in} \cup D_k^{in}$
— **Join:** For two input dataset D_i^{in}, D_k^{in} with schema $rn_i(x_1, \ldots x_n)$ and $rn_k(y_1, \ldots, y_q)$, for a chosen pair of indices ind_i, ind_k compute $D_j^{out} = \{rn'(x_1, \ldots, x_n, y_1, \ldots, y_{ind_k-1}, y_{ind_k+1}, \ldots, y_q) \mid x_{ind_i} = y_{ind_k} \wedge rn_i(x_1, \ldots, x_n) \in D_i^{in} \wedge rn_k(y_1, \ldots y_q) \in D_k^{in}\}$
— **Difference:** For two input dataset D_i^{in}, D_k^{in} with the same schema $rn(x_1, \ldots, x_n)$, compute $D_j^{out} = D_i^{in} \setminus D_k^{in}$

WORKERS OPERATIONS : These are elementary tasks performed by workers to modify the datasets computed so far. These operations may perform non-deterministic choices among possible outputs.

— **Field addition:** Given an input dataset D_i^{in} with schema $rn(x_1, \ldots, x_n)$, a predicate $P(.)$, compute a dataset D_j^{out} with schema $rn'(x_1, \ldots, x_n, x_{n+1})$ such that every tuple $rn(a_1, \ldots, a_n) \in D_i^{in}$ is extended to a tuple $rn(a_1, \ldots, a_n, a_{n+1}) \in D_j^{out}$ such that $P(a_1, \ldots, a_{n+1})$ holds. Note that the value of field x_{n+1} can be chosen *non-deterministically* for each record.
— **Record input:** Given an input dataset D_i^{in} with schema $rn(x_1, \ldots x_n)$, and a predicate P, compute a dataset D_j^{out} on the same schema $rn(x_1, \ldots, x_n)$ with an additional record $rn(y_1, \ldots, y_n)$ such that $P(y_1, \ldots, y_n)$ holds. Note that the value of $y_1, \ldots, y_{n+1}$ can be non-deterministically chosen. Intuitively, P defines the set of possible entries in a dataset.
— **Field update:** For each record $rn(a_1, \ldots, a_n)$ of D_i^{in}, compute a record $rn(b_1, \ldots, b_n)$ in D_j^{out} such that some linear arithmetic predicate $P(a_1, \ldots, a_n, b_1, \ldots, b_n)$ holds. Again, any value for $b_1, \ldots, b_n$ that satisfies P can be chosen.

RECORD TO RECORD ARITHMETIC OPERATIONS (R2R) : For each record $rn_i(a_1, \ldots, a_n)$ of D_i^{in}, compute a record in D_j^{out} of the form $rn_j(b_1, \ldots, b_q)$ such that each $b_k, k \in 1..q$ is a linear combination of $a_1, \ldots, a_n$. For example (Figure 3.4), Consider node n_1 with input dataset with D_1 with relational schema $rn_1(id, c_1, \ldots, c_n)$ where id denotes an item numbers of grocery products and $c_1, \ldots, c_n$ denotes the different cost (raw material cost, labor cost, shipping cost, storage cost, ...). The n_1 node executes R2R operation and returns dataset D_2 with relational schema $rn_2(id, v_1 \ldots, v_n)$ where each v_i denotes different costs beared by the regulators (producer, whole sellers, retailers,...).

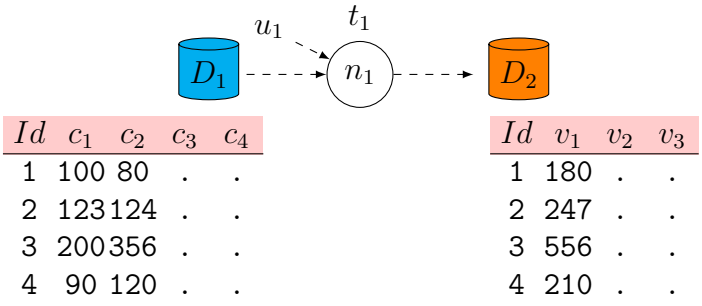


Figure 3.4 – Record to record arithmetic operations.

For example, the field value v_1 denotes the producer cost and is obtained by the linear combinations as $R2R(v_1) = c_1 + c_2$ where c_1 is the raw material cost and c_2 denotes labor cost. Note that here, we consider linear combination of attributes as it gives an effective algorithm to compute the results in polynomial time. We also restrict ourselves to arithmetic over real values. The FO formula with real number and arithmetic operations (Addition, multiplication, ...) with the equality constraints are decidable following the famous Tarski theorem [Tar98]. While for other cases as function symbols, it can lead to undecidable cases (see Richardson's theorem [Ric68]).

These operations can easily define tasks that split a database D_1^{in} in two datasets D_1^{out}, D_2^{out}, one containing records that satisfy some predicate P and the other one records that do not satisfy P. Similarly, one can describe input of record from a worker, operations that assemble datasets originating from different threads. To summarize, when a node n with associated task t with I predecessors is executed, we slightly abuse our notations and write $D_k^{out} = f_{k,t}(D_1^{in}, \ldots, D_I^{in})$ when the task performs a deterministic calculus (SQL-based operations or record to record calculus), and $D_k^{out} \in F_{k,t}(D_1^{in}, \ldots, D_I^{in})$ when the tasks involves non-deterministic choice of a worker and the records in the output dataset should satisfy some properties relation values of record fields in D_k^{out} and values in $D_1^{in}, \ldots, D_I^{in}$. An example is shown in Figure 3.5. The node n_4 receives input dataset D_1, D_2 and D_3 and produces dataset D_4^{out} using a deterministic *union* operations. We use *first-order logic* (FO) to address properties on dataset. Next, we give the operational semantics of complex workflows.

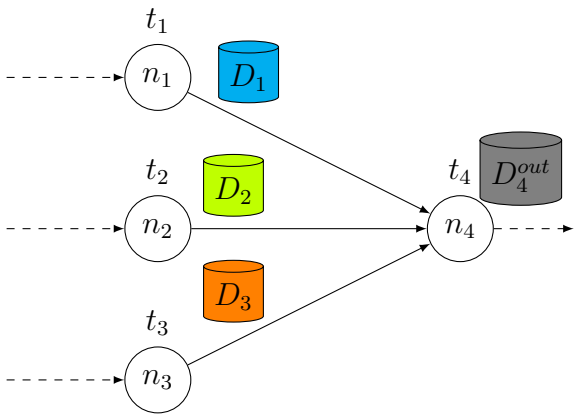


Figure 3.5 – Union of datasets.

3.4.2 Operational Semantics

An **execution** starts from the initial workflow W_0 that is the initial high-level description provided by the client. Without loss of generality, we assume that the client provides an input dataset D_{in} (that can be empty). Executing a complex workflow consists of realizing all its tasks following the order given by the dependency relation $\longrightarrow$ in the orchestration, possibly after some refinement steps. At each step of execution, the remaining part of the workflow to execute, the assignments of tasks to workers, and the data input to tasks are memorized in a *configuration*. Execution steps consist of updating configurations according to operational rules. They assign a task to a competent worker, execute an atomic or automated task (i.e. produce output data from input data), or refine a complex task. Executions end when the remaining workflow to execute contains only the final node n_f.

A *worker assignment* for a workflow $W = (N, \longrightarrow, \lambda)$ is a partial map $wa : N \to \mathcal{U}$ that assigns a worker to a subset of nodes in the workflow. Let $wa(n) = w_i$. If $\lambda(n)$ is a complex task, then there exists a rule $r = (\lambda(n), W) \in \mathcal{R}$ such that $(w_i, r) \in sk$ (worker w_i knows how to refine task $\lambda(n)$). Similarly, if $\lambda(n)$ is an atomic task, then $(w_i, \lambda(n)) \in sk$ (worker w_i has the competences needed to realize $\lambda(n)$). We furthermore require map wa to be injective, i.e. a worker is involved in at most one task. We say that $w_i \in \mathcal{U}$ is *free* if $w_i \notin wa(N)$. If $wa(n)$ is not defined, and w_i is a free worker, $wa \cup \{(n, w_i)\}$ is the map that assigns node n to worker w_i, and is unchanged for other nodes. Similarly, $wa \setminus \{n\}$ is the restriction of wa to $N \setminus \{n\}$. A *data assignment* for a workflow W is a function $Dass : N \to (DB \uplus \{\emptyset\})^*$, that maps nodes in W to sequence of input

datasets. For a node with k predecessors $n_1, \ldots n_k$, we have $Dass(n) = D_1 \ldots D_k$. A dataset D_i can be empty if n_i has not been executed yet, and hence has produced no data. $Dass(n)_{[i/X]}$ is the sequence obtained by replacement of D_i by X in $Dass(n)$.

Definition 22 (Configuration)**.** *A* configuration *of a complex workflow is a triple* $C = (W, wa, Dass)$ *where* W *is a workflow depicting remaining tasks that have to be completed,* wa *is a worker assignment, and* $Dass$ *is a data assignment.*

A complex workflow execution starts from *initial configuration* $C_0 = (W_0, wa_0, Dass_0)$, where wa_0 is the empty map, $Dass_0$ associates dataset D_{in} provided by client to n_{int} and sequences of empty datasets to all other nodes of W_0. A *final configuration* is a configuration $C_f = (W_f, wa_f, Dass_f)$ such that W_f contains only node n_f, wa_f is the empty map, and $Dass_f(n_f)$ represents the dataset that was assembled during the execution of all nodes preceding n_f and has to be returned to the client. The intuitive understanding of this type of configuration is that n_f needs not be executed and simply terminates the workflow by returning final output data. Note that due to data assignment, there can be more than one final configuration, and we denote by $\mathcal{C}_f$ the set of all final configurations.

We define the operational semantics of a complex workflow with four rules that transform a configuration $C = (W, wa, Dass)$ in a successor configuration $C' = (W', wa', Dass')$. Rule 1 defines task assignments to free workers, Rule 2 defines the execution of an atomic task by a worker, Rule 3 defines the execution of an automated task, and Rule 4 formalizes refinement.

Rule 1 (WORKER ASSIGNMENT): A worker $u \in \mathcal{U}$ is assigned to a node n. The rule applies if u is free, has the skills required by $t = \lambda(n)$, if t is not an automated task ($t \notin \mathcal{T}_{aut}$) and if node n is not already assigned to a worker. Note that a worker can be assigned to a node even if the node does not have input data yet, and is not yet executable. This rule only changes the worker assignment part in a configuration.

$$\frac{\begin{array}{l} n \notin Dom(wa) \wedge u \notin coDom(wa) \wedge \lambda(n) \notin \mathcal{T}_{aut} \\ \wedge\, (T_{cs}(t) \subseteq sk(u)) \end{array}}{(W, wa, Dass) \rightarrow (W, wa \cup \{(n, u)\}, Dass)} \tag{3.1}$$

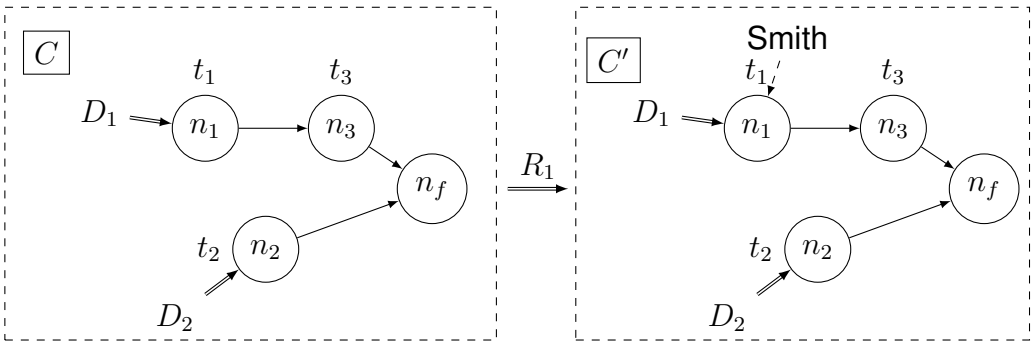


Figure 3.6 – Application of semantic rule R_1.

Consider for instance the application of rule R_1 described in Figure 3.6. Configurations are represented by the contents of dashed rectangles. Workflow nodes are represented by circles, tagged with a task name representing map λ. The dependencies are represented by plain arrows between nodes. Worker assignments are represented by dashed arrows from a worker name u_i to its assigned task. Data assignment are represented by double arrows from a dataset to a node. The left part of Figure 3.6 represents a configuration C with four nodes n_1, n_2, n_3 and n_f. The predecessors of n_1 and n_2 have been executed. Node n_1 represents occurrence of a task of type t_1 and is attached dataset D_1. Let us assume that D_1 is a database containing *bee* pictures, and that task t_1 cannot be automated ($t_1 \notin \mathcal{T}_{aut}$) and consists in tagging these pictures with *bee* names, which requires competences on *bee* species. This is formalized by $T_{cs}(t_1) = \{Bees\}$. Let us assume that worker $Smith$ is currently not assigned any task and has competences on *bees*, i.e. $Bees \in sk(Smith)$. Then rule R_1 applies. The resulting configuration is C' shown at the right of Figure 3.6, where the occurrence of t_1 represented by node n_1 is assigned to worker *Smith*.

Rule 2 (ATOMIC TASK COMPLETION): An atomic task $t = \lambda(n)$ can be executed if node n is *minimal* in workflow W, it is assigned to a worker $u = wa(n)$ and its input data $Dass(n)$ does not contain an empty dataset. Upon completion of task t, worker u publishes the produced data $\mathcal{D}^{out}$ to the succeeding nodes of n in the workflow and becomes available. The rule modifies the workflow part (node n is removed), the worker assignment, and the data assignment (produces new data and is made available to successors of n).

$$\frac{\begin{array}{l} n \in min(W) \wedge \lambda(n) \in \mathcal{T}_{ac} \wedge wa(n) = u \\ \wedge\, Dass(n) \notin DB^*.\emptyset.DB^* \\ \wedge\, \exists \mathcal{D}^{out} = D_1^{out}, \ldots, D_k^{out} \in F_{\lambda(n),u}(Dass(n)), \\ Dass' = Dass \setminus \{(n, Dass(n))\} \cup \\ \quad \{(n_k, Dass(n_k)_{[j/D_k^{out}]}) \mid n_k \in succ(n) \\ \quad \wedge n \text{ is the } j^{th} \text{ predecessor of } n_k\} \end{array}}{(W, wa, Dass) \xrightarrow{\lambda(n)} (W \setminus \{n\}, wa \setminus \{(n,u)\}, Dass')} \tag{3.2}$$

Consider the example of Figure 3.7. We start from a configuration in which worker $Smith$ has to tag *bee* images stored in a dataset D_1. We assume that the relational schema for D_1 is a tuple $R(id, pic)$ where id is a key and pic a picture. We also assume that tags are species names from a finite set of taxons, e.g. $Tax = \{Honeybee, Bumblebee, ..., Unknown\}$. Worker Smith performs the tagging task, which results in a dataset D_3 with relational schema $R'(id, pic, tx)$. One can notice that no information is given of the way worker Smith tags the pictures in D_1, the only insurance is that for every tuple $R(id, pic)$, there exists a tuple $R'(id, pic, tx)$ in D_3 where $tx \in Tax$, i.e.

$$R(id, pic) \in D_1 \implies \exists R'(id, pic, tx) \in D_3 \wedge tx \in Tax$$

Notice that application of this rule may result in several successor configurations, as a human worker can choose non-deterministically any tag for each picture (including wrong answers).

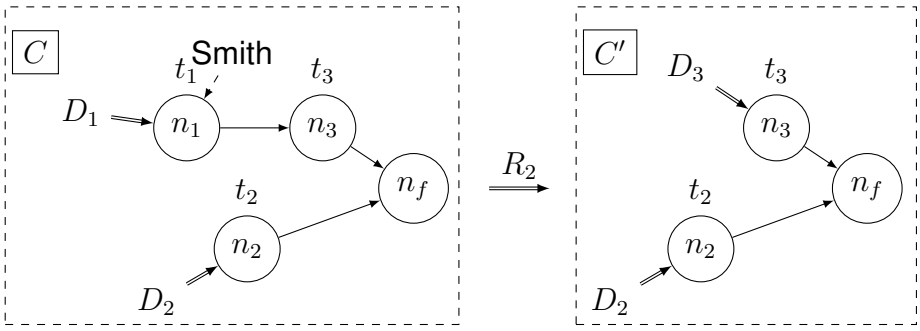


Figure 3.7 – Application of semantic rule R_2.

Rule 3 (AUTOMATED TASK COMPLETION): An automated task $t = \lambda(n)$ can be executed if node n is minimal in the workflow and its input data does not contain an empty dataset. The difference with atomic task completion is that n is not assigned a worker, and the produced outputs are a deterministic function of task inputs. This rule modifies the workflow part (node n is removed), and the data assignment.

$$\frac{\begin{array}{l} n \in min(W) \wedge \lambda(n) \in \mathcal{T}_{aut} \wedge Dass(n) \notin DB^*.\emptyset.DB^* \\ \wedge \mathcal{D}^{out} = f_{\lambda(n),u}(Dass(n)) = D_1^{out}, \ldots, D_k^{out}, \\ Dass' = Dass \setminus \{(n, Dass(n))\} \cup \\ \quad \{(n_k, Dass(n_k)_{[j/D_k^{out}]}) \mid n_k \in succ(n) \\ \quad \wedge n \text{ is the } j^{th} \text{ predecessor of } n_k\} \end{array}}{(W, wa, Dass) \xrightarrow{\lambda(n)} (W \setminus n, wa, Dass')} \tag{3.3}$$

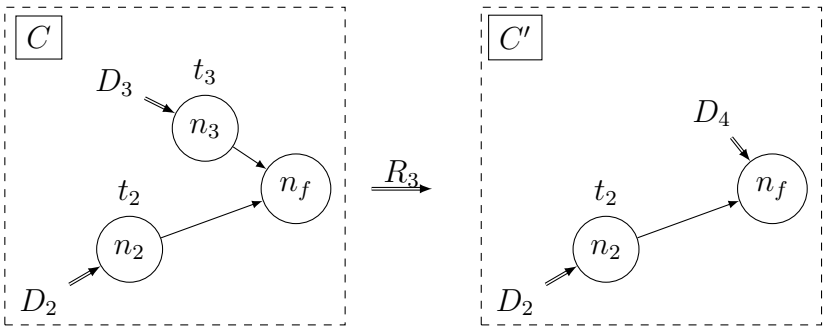


Figure 3.8 – Application of semantic rule R_3.

Consider the example of Figure 3.8. We resume from the situation in Figure 3.7, i.e. with two nodes n_2, n_3 remaining to be executed before n_f, and with a dataset composed of tagged images attached to node n_3. Let us assume that task t_3 is an automated task that consists of pruning out images with tag "unknown". This task can be realized as a projection of D_3 on tuples $R'(id, pic, tx)$ such that $tx \neq$ "$Unknown$". As a result, we obtain a dataset D_4, used as input by node n_f such that $\forall R'(id, pic, tx) \in D_4, tx \neq$ "$unknown$". The projection operation can be realized by simple SQL query as stated in Section 3.4.1.

Rule 4 (COMPLEX TASK REFINEMENT): The refinement of a node n with $t = \lambda(n) \in \mathcal{T}_{cx}$ by worker $u = wa(n)$ uses a refinement rule r such that $r = (t, W_s)$ exists and is listed in the competences of u. The condition for the worker assignment guarantees that refinement is always performed by a competent worker, owning an appropriate

refinement rule to handle a task. Rule 4 refines node n with workflow $W_s = (N_s, \longrightarrow_s, \lambda_s)$ (see Def. 20). Data originally used as input by n become inputs of the source node of W_s. All other newly inserted nodes have empty input datasets. This rule changes the workflow part of configurations and data assignment accordingly.

$$\frac{\begin{array}{l} t = \lambda(n) \in \mathcal{T}_{cx} \wedge \exists u, u = wa(n) \wedge r \in Profile(t,u) \wedge r = (t, W_S) \\ \wedge Dass'(min(W_s)) = Dass(n) \\ \wedge \forall x \in N_s \setminus min(W_s), Dass'(x) = \emptyset^{|Pred(x)|} \\ \wedge wa' = wa \backslash \{(n, wa(n))\} \end{array}}{(W, wa, Dass) \xrightarrow{ref(n)} (W_{[n/W_s]}, wa', Dass')} \tag{3.4}$$

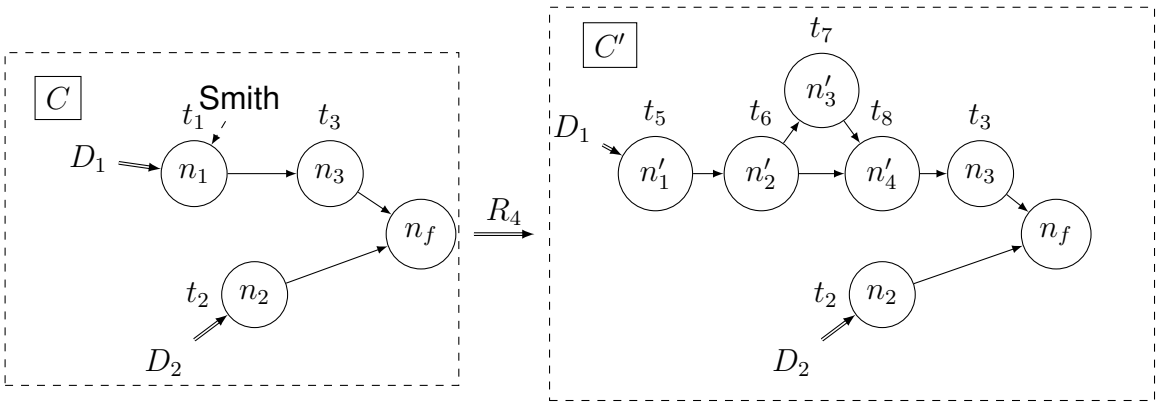


Figure 3.9 – Application of semantic rule R_4.

Consider the example of Figure 3.9. Let us assume that worker *Smith* is assigned task t_1 and that this task is a complex tagging task (for instance workers are asked to find names of rare species). In such a situation, *Smith* can decide to replace the task with a simple single-worker tagging mechanism, or by a more complex workflow, that asks a competent worker to tag pictures, then separates the obtained datasets into pictures with/without the tag “Unknown”, and sends the *Unknown* species to an expert (for instance an entomologist) before aggregating the union of all responses. This refinement leads to a configuration C', shown in the right part of Figure 3.9, where n'_1 is a tagging task, n'_2 is an automated task to split a dataset, n'_3 is a tagging task that requires highly competent workers and n'_4 is an aggregation task. Here the conditions for worker assignment guarantee that refinement is always performed by a competent worker, owning an appropriate refinement rule to handle a task.

Note that the definition of a *complex* task is very subjective and varies from one worker to another. Classifying tasks as complex or not a priori should not be seen as a limitation, as refinement is not mandatory: a worker can replace a node n labeled by task $t \in \mathcal{T}_{cx}$ by another node labeled by an equivalent task $t' \in \mathcal{T}_{ac} \cup \mathcal{T}_{aut}$ if this possibility is allowed by the rules she can apply. This allows to model situations where a worker has the choice to realize a task or refine it when she thinks it is too complex to be handled by a single person. Similarly, the choice of a rewriting rule is a way to implement a choice of a worker or simulate a random environment. But, rewriting allows for recursion, i.e. rewriting of a task t can contain a new occurrence of t (either directly or through successive rewritings).

We say that there exists a *move* from a configuration C to a configuration C', or equivalently that C' is a successor of configuration C and write $C \rightsquigarrow C'$ whenever there exists a rule that transforms C into C'.

Definition 23 (Run)**.** *A run $\rho = C_0.C_1 \dots C_k$ of a complex workflow W is a finite sequence of configurations such that C_0 is the initial configuration of W, and for every $i \in 1 \dots k$, $C_{i-1} \rightsquigarrow C_i$. A run is* maximal *if C_k has no successor. A maximal run is* terminated *iff C_k is a final configuration, and it is* deadlocked *otherwise.*

Runs of a complex workflow are successive rewritings of configurations via rules. Figure 3.10 gives an example of run. The top-left part of the figure is an initial configuration $C_0 = (W_0, wa_0, Dass_0)$ composed of an initial workflow W_0, an empty map wa_0 and a map $Dass_0$ that associates dataset D_{in} to node n_{int}. The top-right part of the figure represents the configuration $C_1 = (W_1, wa_1, Dass_1)$ obtained by assigning worker u_1 for execution of task t_2 attached to node n_2 (Rule 1). The bottom part of the figure represents the configuration C_2 obtained from C_1 when worker u_1 decides to refine task t_2 according refinement rule (t_2, W_{t2}) (Rule 4). Workflow W_{t2} is the part of the Figure 3.10 contained in the Grey square.

We denote by $\mathcal{R}uns(CW, D_{in})$ the set of maximal runs originating from *initial configuration* $C_0 = (W_0, wa_0, Dass_0)$, where $Dass_0$ associates dataset D_{in} to node n_{init}. We denote by $\mathcal{R}each(CW, D_{in})$ the set of configurations that can be reached from C_0. Along a run, the size of the dataset in use can grow unboundedly, and the size of the workflow can also increase, due to the refinement of tasks. Hence, $\mathcal{R}each(CW, D_{in})$ and $\mathcal{R}uns(CW, D_{in})$ need not be finite. Indeed, complex tasks and their refinement can encode unbounded recursive schemes in which workflow parts or datasets grow up to

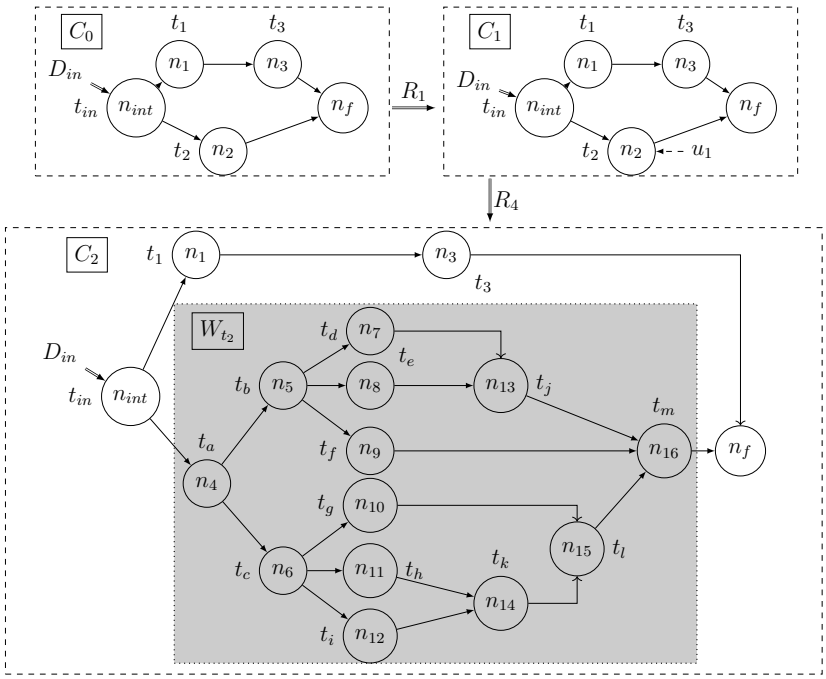


Figure 3.10 – Complex workflow execution. C_0 represents the initial configuration with data D_{in} allocated to node n_{int}. C_1 is the successor of C_0: worker u_1 is allocated to node n_2, and $t_2 = \lambda(n_2)$ is a complex task. C_3 depicts the configuration after refinement of node n_2 by a new workflow W_{t2} (shown in the grey rectangle).

arbitrary sizes. Even when $\mathcal{R}each(CW, D_{in})$ is finite, a complex workflow may exhibit infinite cyclic behaviors. Hence, without restriction, complex workflows define transitions systems of arbitrary size, with growing data or workflow components.

3.5 Conclusion

In this chapter, we defined *complex workflows* for crowdsourcing applications that enable intricate data-centric processes built on higher-order schemes. We presented the operational semantics in the form of rules and rewriting schemes.

However, an operational semantics alone is not enough to guarantee that a workflow provides the desired output required by the client. Complex workflows with rewriting schemes can encode unbounded recursive schemes. In such cases, it becomes crucial to check the termination of workflow along with correctness that guarantees the desired output. In the next chapter, we propose verification schemes for complex workflow and particularly study the termination and correctness properties.

CHAPTER 4

DECIDABILITY

In the previous chapter, we proposed a complex workflow that supports higher-order orchestrations that rewrites a complex task in a new workflow. The complex workflows are equipped with a set of semantic rules (see Section 3.4.2) that describes how to allocate workers, orchestrate and execute tasks. However, a workflow may never reach a final configuration. It can be due to particular data input by workers that cannot be processed properly by the workflow, or to infinite rewriting appearing during the execution. In the first case, the workflow is deadlocked. In the second case, the workflow does not terminate either, this situation is a livelock. Both deadlocks and livelocks prevent termination of a process. However, termination is not the only property to meet the client's requirement. For example, a workflow W may terminate, but with a wrong output, i.e. an output dataset that does not comply with the client's requirement. In such cases, the returned output is of no use to the client. Hence, along with the termination, it is important to guarantee the correctness of the workflow. Given a workflow and a set of workers, we will address the following problems:

— **Universal termination:** Does the workflow terminate for every possible input to the system?

— **Existential termination:** Is there at least one input for which at least one execution of the workflow terminates?

— **Universal correctness:** For a given constraint ψ_0 on the produced results, does the workflow terminates and satisfies ψ_0 for every possible input to the system?

— **Existential correctness:** For a given constraint ψ_0 on the produced results, Is there a particular input and at least one run of execution for which the workflow terminates and satisfies ψ_0?

In this chapter, we study the formal properties of complex workflow: termination and correctness. Here we give the outline of the chapter.

— We first describe the data aspects using first-order logic in Section 4.1.

— We propose solutions to check the existential and universal termination of complex workflows in Section 4.2.

— In Section 4.3, we show how to verify the existential and universal correctness of a complex workflow. We also provide the complete complexity analysis for the termination and correctness problem.
— A use case is provided in Section 4.4.
— We show in Section 4.5, the tool CrowdPlex that implements the complex workflow model and the algorithms to check correctness and termination properties and in the end we conclude in Section 4.6.

4.1 Effective Computation of Weakest Preconditions

Data is central to crowdsourcing systems. Each node of a complex workflow takes some input data that complies with a relational schema for the node, processes it according to the operation attached to this node. More formally, user inputs and automated tasks transform an input dataset into an output dataset. Data processing operations must comply with some FO requirements that constrain the legal outputs produced for some input. This transformation may produce an output in which relational schema is completely different from the schema on the input (for instance when a new field (tag) is added by a crowd worker), or conversely, preserve it (for instance when the operation realized is a selection). The output of a node acts as an input to the successor node(s). Crowdsourcing is used to process data input by a client. One can assume:

— the client has some prior knowledge on the data that can be input to the system.
— the client expects some output that meets certain properties, e.g. the range of answers collected is large enough, the answers are in a particular range, etc.

The most adapted formalism to specify properties of dataset is FO. In the rest of the thesis, we will assume that the FO formulas used are given in Prenex Normal Form (PNF) (see Chapter 2). This results in no loss of generality, as a formula that is not in PNF, can be converted to an equivalent PNF formula. We consider variables that either have finite domains or real valued domains, and predicates specified by simple linear inequalities. In particular, we consider equality of variables, i.e. statements of the form $x_i = x_j$. This class of constraints is well known, and deciding whether there exists an assignment satisfying such predicate can be checked in polynomial time [Kha80]. If we assume that a constraint over n variables is an expression of the form $\bigwedge x_i - x_j \leq c \wedge \bigwedge x_i \leq c$ where x_i, x_j are variables, and c a constant, then constraints can be

encoded as differential bound matrix (DBM), and emptiness of the domain represented by such a DBM where variables takes a real value can be checked in $O(n^3)$. Letting $X_1 = \{x_1, \dots x_k\} \subseteq X$ we write $\forall \vec{X_1}$ instead of $\forall x_1 . \forall x_2 \dots \forall x_k$. Similarly, we write $\exists \vec{X_1}$ instead of $\exists x_1 . \exists x_2 \dots \exists x_k$. Given a FO formula in prenex normal form, without loss of generality we write formulas of the form $\forall \vec{X_1} \exists \vec{X_2} \dots \psi(X)$ or $\exists \vec{X_1} \forall \vec{X_2} \dots \psi(X)$, where $\psi(X)$ is quantifier free matrix, and for every $i \neq j$, $X_i \cap X_j = \emptyset$. Every set of variables X_i is called a *block*. By allowing blocks of arbitrary size, and in particular empty blocks, this notation captures all FO formulas in prenex normal form. We denote by $\phi_{[t_1/t_2]}$ the formula obtained by replacing every instance of term t_1 in ϕ by term t_2.

It is well known that the satisfiability of first-order logic is undecidable in general, but it is decidable for several fragments. Recall that in Chapter 2, we illustrated several decidable FO fragments which include, monadic, BSR, FO2BD, and separated fragments. In our setting, FO formulas contain only Boolean relational predicates (inherited from relational schemas of datasets) and boolean predicates defined as combinations of linear inequalities. The *universal fragment* of FO is the set of formulas of the form $\forall \overrightarrow{X_1}\, \phi$, where ϕ is quantifier-free. Similarly, the existential fragment of FO contains only formulas of the form $\exists \overrightarrow{Y_1} \phi$. Then checking satisfiability of the existential/universal fragment of FO can be done non-deterministically in polynomial time.

Proposition 1. *Let X be a set of variables of the form $X = X_b \uplus X_r$ where variables from X_b take values from finite domains and variables from X_r take values in $\mathbb{R}$. Then, satisfiability of formulas of the form $\phi ::= \exists \overrightarrow{X} . \bigwedge_{i \in 1..I} R_i(X) \wedge \bigwedge_{j \in 1..J} P_j(X)$ is NP-complete.*

Proof. Let us first show that the problem belongs to NP. Let us consider an existential formula $\phi ::= \exists \overrightarrow{X} . \psi$ where ψ contains positive relational statements of the form $\phi_{R+} ::= R_1(X), \dots R_k(X)$, and negative relational statements of the form $\phi_{R-} ::= \neg R_1(X), \dots \neg R_{k'}(X)$, and predicates of the form $P_1(X), \dots P_J(X)$. For each $R_i(x_1, \dots, x_q)$ in ϕ_{R+}, with relational schema rn_i and legal domain Dom_i, we define $Ldom_i$ as the constraint $(x_1, \dots x_q) \in Dom_i$. One can choose non-deterministically in polynomial time a value d_x for each bounded variable x in X_b.

Then one can choose non-deterministically which relational statements and predicate hold, by guessing a truth value $v_j \in \{true, false\}$ for each relation $R_i \in 1..I$ (Resp. predicate $P_j, j \in 1..J$). Now, for each pair of choices where $rn(x_1, \dots x_q)$ holds and $rn(x'_1, \dots x'_q)$ does not, we verify that the designed tuples are disjoint, i.e. that $\neg(x_1 = x'_1 \wedge \dots \wedge x_q = x'_q)$. We call ϕ_{RxR} the formula that is the conjunction of such

negations. The size of ϕ_{RxR} is in $O(r.|\phi^2|)$ where r is the maximal arity in a relational schema of the complex workflow. We can then verify that the guess of truth value for atoms yields satisfaction of ϕ, i.e. check that $\phi'_{[R_i,P_j/true,false,v_j]}$ evaluates to true. In case of positive answer, it suffices to check that with the truth value choosen for atoms, the formula $\phi_{RxR} \wedge \bigwedge_{i \in 1..k} Ldom_i \wedge \bigwedge_{v_i=true} P_i \wedge \bigwedge_{v_i=false} \neg P_i$ is satisfiable, which can be done in polynomial time. Now, for the hardness proof, one can easily encode an SAT problem with a FO formula over boolean variables. Checking satisfiability of a universally quantified formula can be done in the same way, as $\forall X \phi$ is satisfiable iff $\exists X, \neg\phi$ is not. ■

4.1.1 Closure of FO classes

In the next Section 4.2, we study the termination and correctness properties of a workflow. We search for a reachable configuration C_{bad} where the emptiness of a dataset D could stop an execution. Once such a configuration is met, it remains to show that the statement $D = \emptyset$ is compatible with the data operations as insertion, projection, union of datasets, etc. performed during the execution before reaching C_{bad}. We compute backward the weakest preconditions (see def. 16) ensuring $D = \emptyset$ along the followed run and check that each condition is satisfiable. In this section, we first show that some decidable fragments of first-order are closed under the calculus of weakest precondition. The proofs and definition act as a base to further study the properties of the workflow. The weakest precondition in a workflow is defined as follows.

Definition 24. *Let $C \longrightarrow C'$ be a move from configuration C to C' of a complex workflow. Let m be the nature of this move (an automated task realization, a worker assignment, a refinement,...). We denote by $wp[m]\psi$ the* weakest precondition *required for C such that ψ holds in C' after move m.*

As shown in Chapter 2, weakest precondition were introduced in [Dij75] as a way to prove the correctness of programs. Calculus of weakest precondition was also proposed to verify web applications with embedded SQL [Itz+17]. Similarly, we can show an effective procedure to compute the weakest preconditions for each operation in a complex workflow.

A technical lemma for weakest precondition. Here, we illustrate the derivation of the weakest precondition for the Selection operation. The *selection* operation returns a set of records from a table. We give below the lemma to derive the weakest precondition for a task that performs a selection of records that satisfy some predicate in a dataset.

Lemma 1 (Weakest precondition for Selection of records)**.** *Let φ be a FO formula, and m be a move that selects records that satisfy a predicate P from datasets. Then one can effectively compute an FO formula $\psi = wp[m]\varphi$. Moreover, if φ is in $\forall$FO (resp. $\exists$FO, BSR, SF) and P is an arithmetic/boolean predicate then ψ is also in $\forall$FO (resp. $\exists$FO, BSR, SF).*

Proof. Let $D'_1, \dots D'_j, \dots D'_k \models \varphi$, and let D'_j be a dataset with relational schema $rs = (rn, A)$ obtained by selection of records from an input dataset D_i with relational schema $rs(rn, A)$. One can notice that selection keeps the same relational schema, and in particular the same set of attributes $A = (a_1, \dots a_k)$. We assume that selected satisfy some predicate $P(v_1, \dots v_k)$ that constrain the values of a record (but do not address properties of two or more records of D_i). That is, the records selected from D_i by P are records that satisfy $\psi_{sel} = \exists v_1, \dots v_k, rn(v_1, \dots v_p) \wedge P(v_1, \dots, v_k)$. We want to compute $\psi = wp[Selection(\psi_{sel})]\ \varphi$.

Formula φ is a formula of the form $\alpha(\overrightarrow{X}).\phi$, where $\alpha(\overrightarrow{X})$ is a prefix. It contains K_{rn} subformulas of the form $rn(w_i, \dots w_{i+k})$ or $\neg rn(w_i, \dots, w_{i+k})$ and we assume without loss of generality that these subformulas are over disjoint sets of variables (one can add new variables and equalities if this is not the case). Let $\phi_{rn,1}, \dots \phi_{rn,K_{rn}}$ be the subformulas of ϕ addressing tuples with relational schema rs. For $i \in 1 \dots K_{rn}$, we let $\phi^P_{rn,i}$ denote the formula $rn(w_i, \dots w_{i+k}) \wedge P(w_i, \dots w_{i+k})$ if $\phi_{rn,i}$ is in positive form and $\neg(rn(w_i, \dots w_{i+k}) \wedge P(w_i, \dots w_{i+k}))$ otherwise. Let us denote by $\phi_{[\{\varphi_{rn,i}\}|\{\varphi^P_{rn,i}\}]}$ the formula ϕ where every $\phi_{rn,i}$ is replaced by $\phi^P_{rn,i}$. The weakest precondition on $D'_1, \dots D_i$, $\dots, D'_k$ for a selection operation with predicate P is defined as

$$\psi = wp[Selection(\psi_{sel})]\varphi = \alpha(\overrightarrow{X}).\phi_{[\{\phi_{rn,i}\}|\{\phi^P_{rn,i}\}]}$$

Last, one can notice that transforming $\alpha(\overrightarrow{X}).\phi$ into $\alpha(\overrightarrow{X}).\phi_{[\{\phi_{rn,i}\}|\{\phi^P_{rn,i}\}]}$ does not introduce new variables, and preserve the prefix of the formula. As φ and ψ start with the same prefix $\alpha(\overrightarrow{X})$, we can claim that φ and ψ are in the same fragment of FO. ■

One can notice that the weakest precondition is a rather syntactic transformation, that replaces atoms of the form $rn(x_1, \dots x_k)$ by $rn(x_1, \dots x_k) \wedge P(x_1, \dots x_k)$. If $x_1, \dots, x_k$ are all existentially quantified variables (resp. all universally quantified variables) in φ, then they remain existentially (resp universally quantified) in ψ. Hence, if φ is in $\exists$ FO, $\forall$FO, BSR, SF-FO then so is $wp[Selection(\psi_{sel})]\varphi$. The same remark applies to all other types of moves except for automated tasks that perform datasets difference,

which may introduce quantifiers alternations. A complete proof is available in [Bou+19]. In [Bou+19], we also show that the size of a weakest precondition ψ for a given φ and a maximal arity r in relation is $O(r.|\psi|)$.

Proposition 1: *Let CW be a complex workflow, r be the maximal arity of relational schemas in CW and φ be a FO formula. Then for any move m of CW, $wp[m]\varphi$ is an effectively computable FO formula, and is of size in $O(r.|\psi|)$.*

Proof. The effect of moves on the contents of datasets can be described as a sequential composition of basic operations that are projection, selection, insertion of records or fields, difference, union or join. In lemma 1, we showed that one can effectively compute the weakest precondition for selection operation. Now, we first just sketch the proof for the rest of the data operations and give the syntactic transformation used to compute the weakest preconditions. Let $D_1, \dots D_k$ be the datasets that appear in ψ.

If some D_i is obtained as a selection of records from a dataset D'_i, then D_i contains only records of D'_i satisfying some predicate P. The precondition will hence be obtained by a simple replacement in ψ of any statement of the form $rn(\overrightarrow{X}) \in D_i$ by $rn(\overrightarrow{X}) \in D_i \wedge P(\overrightarrow{X})$.

If D_i is obtained after insertion of a fresh record in some dataset D'_i then every statement $rn(\overrightarrow{X}) \in D_i$ can be replaced by a subformula $(rn(\overrightarrow{X}) \in D'_i \vee Dom_i(\overrightarrow{X}))$ where $Dom_i(\overrightarrow{X})$ represents constraints on legal values of inputs in a dataset with the same relational schema as D_i. Note that $Dom_i(\overrightarrow{X})$ is a quantifier free boolean combination of predicates.

Similarly, if some dataset D_i is obtained as the union of two datasets D_1, D_2, the precondition for ψ should consider that tuples that satisfy $rn(\overrightarrow{X}) \in D_i$ belong to D_1 or D_2. We simply replace atoms of the form $rn(\overrightarrow{X}) \in D_i$ by the disjunction $rn(\overrightarrow{X}) \in D_1 \vee rn(\overrightarrow{X}) \in D_2$. We also replace negative statements $rn(\overrightarrow{X}) \notin D_i$ by conjunction $rn(\overrightarrow{X}) \notin D_1 \wedge rn(\overrightarrow{X}) \notin D_2$. The size of the obtained formula is hence in $O(2.|\psi|)$.

If some D_i is obtained by creation of a new field for each record in dataset D'_i, then relational statement $rn(\overrightarrow{X}) \in D_i$ is replaced by another statement $rn(\overrightarrow{Y}) \in D'_i \wedge P'(\overrightarrow{Y})$ where $\overrightarrow{Y}$ is a subset of $\overrightarrow{X}$, and $P'(\overrightarrow{Y})$ is a quantifier free predicate indicating constraint on $\overrightarrow{Y}$ obtained after variable elimination when $\overrightarrow{X}$ takes legal values imposed by relational schema of D_i (i.e. it satisfies $Ldom_i$ –see proof of Proposition 1–) and satisfies the constraints of relational schema of D'_i. As we assume that arithmetic predicates are simple (two-dimensional) inequalities, the size of P' is not larger than that of

$Ldom_i$. The weakest precondition can hence double the number of atoms, but keeps the same bound on the number of variables used.

For (binary) joins, relation of the form $rs(\overrightarrow{X}_i)$ are transformed in statements of the form $rs(\overrightarrow{Y_i}) \wedge rs(\overrightarrow{Z_i}) \wedge y_1 = z_1$, where $y_1 \in \overrightarrow{Y_i}$ and $z_1 \in \overrightarrow{Z_i}$. As every relational statement can be replaced and hence create new variables, this calculus of a weakest precondition gives a formula of size in $O(2.|\psi|)$.

If some D_i is obtained by difference of two datasets, D_2 from D_1, the precondition for ψ should consider that tuples that satisfy $rn(\overrightarrow{X}) \in D_i$ belong to D_1 and $rn(\overrightarrow{X}) \in D_i$ should not be present in D_2. We replace atoms of the form $rs(\overrightarrow{X}) \in D_i$ by the conjunction $rn(\overrightarrow{X}) \in D_1 \wedge rn(\overrightarrow{X}) \notin D_2$. We also replace negative statements $rn(\overrightarrow{X}) \notin D_i$ by disjunction $rn(\overrightarrow{X}) \notin D_1 \vee rn(\overrightarrow{X}) \in D_2$. The size of the obtained formula is hence in $O(2.|\psi|)$.

Last, if some D_i is a projection of some dataset D'_i on a subset of its field, then the weakest precondition for ψ may multiply the number of variables in use by r, hence gives a formula of size in $O(r.|\psi|)$. ■

We illustrate a *Join* operation with an example shown in Figure 4.1. Let D_1 be the dataset produced as output of node n_1, with relational schema $rs_1(x_1, x_2, x_3)$. Let D_2 be the dataset produced as output of node n_2 with relational schema $rs_2(y_1, y_2)$. Last, let D_3 be the dataset obtained by a *join* operation identifying x_1 and y_1 in record originating from D_1 and D_2. Assume that formula ϕ is of the form $\exists z_1, z_2, z_3, z_4, rn(z_1, z_2, z_3, z_4) \in D_3 \wedge P(z_1, z_2, z_3, z_4)$. Then the weakest precondition for ϕ is $wp[join]\phi ::= \exists z_1, z_2, z_3, z_4, y_1, rn_1(z_1, z_2, z_3) \in D_1 \wedge rn_2(y_1, z_4) \in D_2 \wedge (z_1 = y_1) \wedge P(z_1, z_2, z_3, z_4)$.

Next, we show in Section 4.2 that many properties of complex workflows are decidable with some restrictions on recursion and when difference is not used in automated

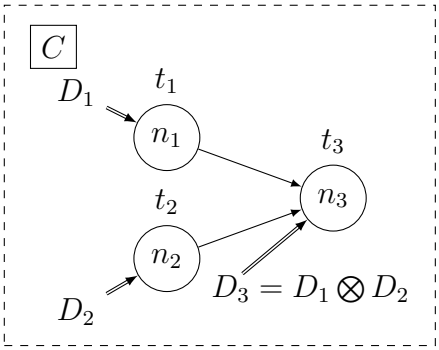


Figure 4.1 – Join operation Example.

tasks of the workflow. Prop. 1 does not need such assumption. The weakest preconditions calculus is rather syntactic and is effective for any FO formula and any move. Now, if automated tasks use *difference*, universal quantifiers can be introduced in existential blocks, leading to the weakest preconditions in a FO fragment where satisfiability is undecidable. Interestingly, if automated tasks do not use *difference*, the weakest precondition is mainly obtained by syntactic replacement of relational statements and changes of variables. It can increase the number of variables, but it does not change the number of quantifier blocks nor their ordering and does not introduce new quantifiers when replacing an atom. We hence easily obtain:

Corollary 1. *The existential, universal, BSR and SF fragments of FO are closed under calculus of the weakest precondition if tasks do not use difference.*

4.2 Termination of Complex Workflows

In this section, we address the questions of existential and universal termination. Universal termination provides a guarantee that a complex workflow will terminate and return a result for any input. This can be seen as a termination guarantee. Existential termination can be considered as a sanity check: a complex workflow that has no valid execution never terminates, regardless of input data, and should hence be considered as ill-formed. Universal correctness guarantees that a workflow terminates and that it computes data that conforms to the client's requirement. Existential correctness is also a sanity check, showing that a workflow is able to produce correct data for at least one of its executions. We first show that existential termination is undecidable, regardless of the inputs specified for a workflow. We then show that decidability and complexity of universal termination depend on the power allowed to specify inputs of the workflow. We first define the termination as follows.

Definition 25 (Termination)**.** *Let CW be a complex workflow, D_{in} be an initial dataset, $\mathcal{D}_{in}$ be a set of datasets.*

- *CW terminates existentially on input D_{in} iff there exists a run in $\mathcal{R}uns(CW, D_{in})$ that is terminated. Similarly, CW* terminates existentially *on $\mathcal{D}_{in}$ iff some run of CW* terminates *for an input $D_{in} \in \mathcal{D}_{in}$.*
- *CW terminates universally on input D_{in} iff all runs in $\mathcal{R}uns(CW, D_{in})$ are terminated. CW terminates universally on input set $\mathcal{D}_{in}$ iff CW terminates universally on every input $D_{in} \in \mathcal{D}_{in}$*

When addressing termination for a set of inputs, we describe $\mathcal{D}_{in}$ symbolically with a decidable fragment of FO ($\forall$FO, $\exists$FO, BSR, or SF-FO). Complex workflows are Turing powerful. The proof of undecidability comes from a reduction from a two-counters machine. A complete proof (an encoding of a counter machine) can be found in Appendix A.1.

Theorem 1. *Existential termination of complex workflows is undecidable.*

An interesting point is that undecidability does not rely on arguments based on the precise contents of datasets (that are ignored by semantic rules). Indeed, the execution of tasks only requires non-empty input datasets. Undecidability holds as soon as higher-order operations (semantic rule $R4$) are used. Universal termination is somehow an easier problem than existential termination.

We next show that it is decidable for many cases and in particular when the datasets used as inputs of a complex workflow are explicitly given or are specified in a decidable fragment of FO. Precisely, we address operations, constraints and data operations on $\mathcal{D}_{in}$ symbolically with a decidable fragment of FO ($\forall$FO, $\exists$FO, BSR, or SF-FO). We proceed in several steps. We first define symbolic configurations, i.e. descriptions of the workflow part of configurations decorated with relational schemas depicting data available as an input of tasks. We define a successor relation for symbolic configurations. We then identify the class of non-recursive complex workflows, in which the length of executions is bounded by some value $K_{\mathcal{T}_{cx}}$. We show that for given finite symbolic execution ρ^S, and a description of inputs, one can check whether there exists an execution ρ that coincides with ρ_S. This proof builds on the effectiveness of the calculus of weakest preconditions along a particular run (see Prop. 1). Then, showing that a complex workflow does not terminate amounts to proving that it is not recursion-free, or that it has a finite symbolic run which preconditions allow a deadlock. We start by defining the symbolic configurations.

Definition 26 (Symbolic configuration)**.** *Let $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ be a complex workflow with database schema DB. A* symbolic configuration *of CW is a triple $C^S = (W, wa, Dass^S)$ where $W = (N, \rightarrow, \lambda)$ is a workflow, $wa : N \rightarrow \mathcal{U}$ assigns workers to nodes, and $Dass^S : N \rightarrow (DB)^*$ associates a list of relational schemas to nodes of the workflow.*

Symbolic configuration describes the status of workflow execution as in standard configurations (see def. 22) but leaves the data part (the actual contents of dataset)

unspecified. For every node n that is minimal in W, $Dass^S(n) = rs_1, \ldots rs_k$ implies that task attached to node n takes as input dataset $D_1 \ldots D_k$ where each D_i conforms to relational schema rs_i. For a given symbolic configuration, we find all rules that apply (there is only a finite number of worker assignments or task executions) and compute successor symbolic configurations. From a symbolic configuration, that does not describe the exact contents of datasets in use, we compute the effects of the application of a particular rule, i.e. compute symbolic descriptions of configurations that appear after a move. A symbolic configuration $C_j^S = (W_j, wa_j, Dass_j^S)$ is the *successor* of a symbolic configuration $C_i^S = (W_i, wa_i, Dass_i^S)$ iff one of the following situation holds:

1. There exists a worker $u \in \mathcal{U}$ and a node $n \in W_i$ such that $wa_i^{-1}(u) = \emptyset$, $wa_i(n) = \emptyset$, $\exists(u, \lambda(n)) \in sk$ or $(u, r) = (\lambda(n), W_t)) \in sk$, and $W_j = W_i$, $Dass^S = Dass^S$ and $wa_j = wa_i \uplus \{(n, u)\}$. The situation corresponds to worker assignment to a task (see Rule 3.1).
2. There exists $n \in min(W_i)$ such that $t = \lambda(n)$ is an automated task manipulating datasets $D_1, \ldots D_q$, n has k successors $n_1, \ldots n_k$, $W_j = W_i \setminus \{n\}$, $Dass^S$ assigns to each successor $n_j, j \in 1 \ldots k$ the relational schema rs_j^{out} and $wa_j = wa_i$. The situation corresponds to an application of an automated task completion Rule 3.3.
3. There exists $n \in min(W_i)$ such that $t = \lambda(n)$ is an atomic task manipulating datasets $D_1, \ldots D_q$, with and n has k successors $n_1, \ldots n_k$, $W_j = W_i \setminus \{n\}$, $Dass^S$ assigns to each successor $n_j, j \in 1 \ldots k$ the relational schema rs_j^{out}, and $wa_j = wa_i \setminus \{(n, wa_i(n))\}$. The situation corresponds to an application of Rule 3.2 where a worker executes an atomic task and returns the data to successor nodes.
4. There exists $n \in W_i$, $\lambda(n)$ is a complex task, and $wa(n) = u$, W_j is the workflow obtained by replacement of n in W_i by a workflow W^{new} such that $r = (\lambda(n), W^{new}) \in \mathcal{R}$, and $(u, r) \in sk$. $Dass^S$ assigns to the copy of minimal node n_j of W^{new} the relational schemas in $Dass_i^S(n)$, and $wa_j = wa_i \setminus \{(n, u)\}$. It corresponds to the refinement of a node in a workflow and is obtained by application of Rule 3.4.

Now, we define deadlocks and potential deadlock in symbolic workflow executions.

Definition 27 (Deadlocks, Potential deadlocks)**.** *A symbolic configuration* $C^S = (W, wa, Dass^S)$ *is* final *if* W *consists of a single node* n_f*. It is a deadlock if it has no successor. It is a potential deadlock iff a task execution can occur from this configuration, i.e. there exists* $n, \in \min(W)$ *such that* $\lambda(n)$ *is an automated or atomic task.*

A deadlocked symbolic configuration represents a situation where progress is blocked due to the shortage of competent workers to execute tasks. A potential deadlock is a symbolic configuration C^S where empty datasets may stop an execution. This deadlock is *potential* because $Dass^S$ does not indicate whether a particular dataset D_i is empty. Note however as soon as a dataset D_i used as input to a task associated with a minimal node $n \in W$ is empty, deadlock becomes unavoidable. Indeed, as n is minimal, no new data can appear in D_i in the next moves. Hence it is unavoidable to reach a deadlock node as soon as $D_i = \emptyset$. We show that one can decide whether a potential deadlock situation in C^S represents a real and reachable deadlock, by considering how the contents of dataset D_i is forged along the execution leading to C^S.

Definition 28 (Symbolic run)**.** *A* symbolic run *is a sequence* $\rho^S = C_0^S \xrightarrow{m_1} C_1^S \xrightarrow{m_2} \ldots \xrightarrow{m_k} C_k^S$ *where each* C_i^S *is a symbolic configuration,* C_{i+1} *is a successor of* C_i*, and* $C_0^S = (W_0, wa_0, Dass^S)$ *where* $W_0,\ wa_0$ *have the same meaning as for configurations, and* $Dass_0^S$ *associates to the minimal node* n_0 *in* W_0 *the relational schema of* $Dass_0(n_0)$*.*

One can associate to every execution of a complex workflow $\rho = C_0 \xrightarrow{m_1} C_1 \ldots \xrightarrow{m_k} C_k$ a symbolic execution $\rho^S = C_0^S \xrightarrow{m_1} C_1^S \xrightarrow{m_2} \ldots \xrightarrow{m_k} C_k^S$ called its *signature* by replacing data assignment in each $C_i = (W_i, wa_i, Dass_i)$ by a function from each node n to the relational schemas of the datasets in $Dass_i(n)$. It is not true, however, that every symbolic execution is the signature of an execution of CW, as some moves might not be allowed when a dataset is empty (this can occur for instance when datasets are split, or after a selection). A natural question when considering a symbolic execution ρ^S is whether it is the signature of an actual run of CW. The proposition below shows that the decidability of this question depends on assumptions on input datasets.

Proposition 2. *Let* CW *be a complex workflow,* D_{in} *be a dataset,* $\mathcal{D}_{in}$ *be a FO formula with* n_{in} *variables, and* $\rho^S = C_0^S \ldots C_i^S$ *be a symbolic run. If tasks do not use SQL difference, then deciding if there exists a run* ρ *with input dataset* D_{in} *and signature* ρ^S *is in* $2EXPTIME$*. Checking if there exists a run* ρ *of* CW *and an input dataset* D_{in} *that satisfies* $\mathcal{D}_{in}$ *with signature* ρ^S *is undecidable in general. If tasks do not use SQL difference, then it is in*

- — $2EXPTIME$ *if* $\mathcal{D}_{in}$ *is in* $\exists$*FO.*
- — $3EXPTIME$ *if* $\mathcal{D}_{in}$ *is in* $\forall$*FO or BSR-FO.*
- — $n_{in} - foldEXPTIME$ *where* n_{in} *is the size of the formula describing* $\mathcal{D}_{in}$ *if* $\mathcal{D}_{in}$ *is in SF-FO*

Proof. We check the feasibility of ρ^S, that is that is we check existence of a run $C_0, C_1, \ldots C_{i-1}, C_i$. We start from C_i^S and check backward that the conditions for existence of a configuration C_k are met when it is proved that a configuration C_{k+1} exists. This amounts to computing a sequence of weakest preconditions. First notice that the actual run with signature ρ^S performs the same sequence of moves as in ρ^S and that the question of the existence of a run ρ with signature ρ^S only needs to verify satisfiability of constraints on data computed at each step of this run, not the sequence of moves along ρ. Second, one can notice that if ρ^S contains a deadlock, it is necessarily the last symbolic configuration of the run, as for every symbolic configuration from C_0^S up to C_{i-1}^S we are able to find a successor configuration. So one needs not to check the existence of a deadlock separately when checking the feasibility of ρ^S, and we mainly have to check for the emptiness of datasets for configurations that are potential deadlocks. A third remark is that semantic rules that affect workers to tasks or perform a refinement do not consider data contents. Hence, if the move from C_{i-1} to C_i is a worker assignment or a refinement, then it is necessarily feasible as long as C_{i-1} is reachable. The only cases where data can affect the execution of a step along a run is when an automated task or an atomic task has to process empty data. For each of these steps, one has to check that the inputs of an executed task t are not empty, i.e. suppose that $D_1 \neq \emptyset \wedge \cdots \wedge D_k \neq \emptyset$ for some datasets $D_1 \ldots D_k$ used by t. Non-emptiness of a dataset D_k is simply encoded by the ∃FO formula $\exists \overrightarrow{X}, rn(\overrightarrow{X}) \in D_k$.

Non-emptiness of a dataset D_k at some configuration C_j is a property that depends on properties of previous steps in the execution. For instance, if the move from C_{j-1} to C_j realizes the projection of a dataset, i.e. filters records in a dataset D'_k to keep only those that satisfy some predicate P, then the precondition that must hold at C_{j-1} is $\psi_{j-1} ::= \exists \overrightarrow{X}, rn(\overrightarrow{X}) \wedge P(\overrightarrow{X})$. We have seen in Proposition 1 that , if the move $C_{j-1} \xrightarrow{m_{i-1}} C_{i-1}$ is a transformation of records, a transformation of some dataset, the formula ψ_{j-1} is effectively computable. Further, if tasks do not use SQL difference, the weakest precondition of an existential FO formula is an existential FO formula. This generalizes to the whole signature. Let ψ_k be a ∃FO formula that has to be satisfied by configuration C_k in a run compatible with signature ρ^S. There exists a sequence of moves $C_0 \xrightarrow{m_1} C_1 \ldots \xrightarrow{m_k} C_k$ iff the sequence $C_0 \xrightarrow{m_1} C_1 \ldots \xrightarrow{m_{k-1}} C_{k-1}$ ends in a configuration C_{k-1} such that $C_{k-1} \models wp[m_k]\psi_k$ (by definition of weakest precondition). One can decide whether $wp[m_k]\psi_k$ is satisfiable, as ψ_k is in ∃FO, and by Prop. 1, $wp[m_k]\psi_k$ is effectively computable and in the ∃FO fragment. If $wp[m_k]\psi_k$ is not satisfiable, then the move from

C_{k-1} to C_k *always* ends with datasets that do not fulfill ψ_k. If $wp[m_k]\psi_k$ is satisfiable, then the runs that reach C_{k-1} are realizable only if we assume that several datasets (used as input of some task realized at step k) are non-empty at stage $k-1$. We then have to add statements of the form $D_i \neq \emptyset$ to obtain a formula that should hold at step $k-1$, and get a formula of the form $\psi_{k-1} ::= wp[m_k]\psi_k \wedge D_1 \neq \emptyset \wedge \cdots \wedge D_m \neq \emptyset$. This adds only an existential conjunction, so ψ_{k-1} is also in $\exists$FO. One can start from $\psi_i ::= true$ and build inductively all weakest preconditions $\psi_{i-1}, \psi_{i-2}, \ldots, \psi_0$ that have to be satisfied respectively by configurations $C_{i-1}, \ldots, C_0$ so that an actual run of the complex workflow with signature $C_0^S \ldots C_i^S$ exists. If any of these preconditions is unsatisfiable, then there exists no run with signature $C_0^S \ldots C_k^S$ leading to a configuration C_i compatible with C_i^S, and hence ρ^S is not the signature of an actual run of CW. The size of ψ_0 is in $O(i.r^i)$. Indeed, we add obligations to prove non-emptiness at each step k, but proving satisfiability of $\exists \overrightarrow{X}, \phi(\overrightarrow{X}) \wedge \exists \overrightarrow{X}, \phi(\overrightarrow{X})$ amounts to checking separately satisfiability of $\exists \overrightarrow{X}, \phi(\overrightarrow{X})$ and $\exists \overrightarrow{Y}, \phi(\overrightarrow{Y})$. According to Prop. 1, the size of $wp[m_k]\psi$ is in $O(r.|\psi|)$, where r is the maximal arity of relational schemas of the complex workflow. So one can check separately satisfiability of $\exists \overrightarrow{X}, \phi(\overrightarrow{X})$ and $D_x \neq \emptyset$, and maintain a series of $O(i)$ formulas of total size in $O(i.r^i)$. Hence, as ψ_0 is still in the existential fragment of FO, and as checking satisfiability of an existential FO formula is in $EXPTIME$ in the size of the formula (by proposition 1), checking all preconditions for a run of size k compatible with ρ^S can have a complexity that is in $O(2^{r^i})$.

Assume that $\psi_{k-1}, \psi_{k-2}, \ldots \psi_0$ are satisfiable. It remains to show that the input(s) of the complex workflow satisfy the weakest precondition for the execution of ρ^S, i.e. satisfy ψ_0. Then, when the input is a single dataset D_{in}, it remains to check that $D_{in} \models \psi_0$ to guarantee the existence of a run with signature $C_0^S \ldots C_i^S$ that starts with input data D_{in} and leads to a configuration C_k. This is a standard model checking question, which can be solved in $O(|D_{in}|^{|\psi_0|})$, that is in $O(|D_{in}|^{r^k})$. As the complexity of checking satisfiability of weakest preconditions $\psi_k \ldots \psi_0$ is already in $2EXPTIME$, the overall complexity is in $2EXPTIME$.

Similarly, if $\mathcal{D}_{in}$ is a FO formula, the complexity depends on the considered fragment used to specify $\mathcal{D}_{in}$. In general, if $\mathcal{D}_{in}$ is given as a FO formula, it is undecidable if $\mathcal{D}_{in} \wedge \psi_0$ is satisfiable. If $\mathcal{D}_{in}$ is an existential formula, then the complexity is exponential in the size of $\mathcal{D}_{in}$ and also exponential in the size of ψ_0. Assuming that $|\mathcal{D}_{in}| \leq 2^k$ we have a $2EXPTIME$ complexity. If $\mathcal{D}_{in}$ is in the BSR fragment, then checking satisfiability of $\mathcal{D}_{in}$ is in $NEXPTIME$ [Lew80], and so the overall complexity needed to check the

existence of a run with signature ρ^S from a dataset in $\mathcal{D}_{in}$ is in 2EXPTIME w.r.t the size of $\mathcal{D}_{in} \wedge \psi_0$, and hence 3EXPTIME. If $\mathcal{D}_{in}$ is a universal formula then we can use standard mini-scoping rules (see in Chapter 2) to transform $\mathcal{D}_{in} \wedge \psi_0$ in a formula in the BSR fragment, yielding again a 3EXPTIME complexity. Last if $\mathcal{D}_{in}$ is in the separated fragment of FO, then checking its satisfiability is n_{in}-fold exponential in the size of the formula depicting $\mathcal{D}_{in}$, so the overall process of checking realizability of ρ^S has an n_{in}-fold exponential complexity. ■

An execution $\rho = C_0 \dots C_k$ of a complex workflow terminates if the reached configuration is of the form $C_k = (W_f, wa_f, Dass_f)$ where W_f contains only the final node of a workflow. Checking termination hence amounts to checking whether one can reach such a configuration. A run that does not terminate is a run that either ends in a configuration that is not final and from which no rule can be applied, or an infinite run. A move from C_i to C_{i+1} leaves the number of nodes unchanged (application of worker assignment rule R_1), decreases the number of nodes (execution of an atomic task (R2), or an automated task (R3)), or refines a node in W_i (application of rule R_4). Only in this latter case, the number of nodes may increase. The set of possible transformations of W and wa occurring from C is bounded. Further, semantic rule $R4$ is the only rule that creates new nodes in the workflow part of a configuration. So when the number of occurrences of rule $R4$ in a run is bounded, the number of applications of rules $R1$, $R2$, $R3$ and hence the size of (symbolic) executions is also bounded. Complex workflows that can exhibit infinite runs are hence specification with recursive rewriting schemes.

Definition 29. *Let t be a complex task. We denote by $Rep(t)$ the task names that can appear when refining task t, i.e. $Rep(t) = \{t' \mid \exists u, W, (u,t) \in sk \wedge (t,W) \in \mathcal{R} \wedge t' \in \lambda(N_W)\}$. The* rewriting graph *of a complex workflow $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ is the graph $RG(CW) = (\mathcal{T}_{cx}, \longrightarrow_R)$ where $(t_1, t_2) \in \longrightarrow_R$ if $t_2 \in Rep(t_1)$. Then CW is* recursion-free *if there is no cycle of $RG(CW)$ that is accessible from a task appearing in W_0.*

When a complex workflow is not recursion free, then some executions may exhibit infinite behaviors in which some task t_i is refined infinitely often. Let us consider an example. Let us consider a workflow such that $n_0 \to n_1 \to n_f$ shown in Figure 4.2 .

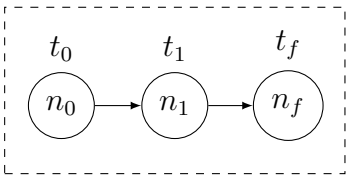


Figure 4.2 – A complex workflow.

Consider the rewriting rules $t_1 = \{t_2 \to t_3\}$, $t_2 = \{t_4 \to t_5\}$ and $t_4 = \{t_1 \to t_6\}$. In this case the rewriting graph of a complex workflow is shown in Figure 4.3. Such an infinite rewriting loop can contain a deadlock. In this case, the complex workflow does not terminate. If this infinite rewriting loop does not contain deadlocks, the complex workflow execution will never reach a final configuration in which all tasks have been executed. We claim without proof the following property:

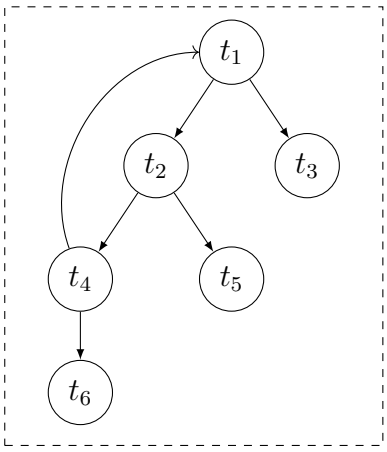


Figure 4.3 – Rewriting graph for the complex workflow shown in Figure 4.2.

Proposition 3. *Complex workflows that are not recursion free have non-terminating executions.*

Proposition 4. *Let $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ be a complex workflow. One can decide if CW is recursion free in $O(|\mathcal{T}_{cx}^2| + |\mathcal{R}|)$.*

Proof. Building $RG(CW)$ can be done in $O(|\mathcal{R}|)$. Checking the existence of a cycle in $RG(CW)$ that is accessible from some task in W_0 can be done in polynomial time in the size of $RG(CW)$, for instance using a DFS algorithm, that runs in time in $O(|\mathcal{T}_{cx}|^2)$. ■

In executions of recursion free CWs, a particular task t can be replaced by a workflow that contains several tasks $t_1, \ldots t_k$ that differ from t. Then, each t_i can be replaced by workflows combining other tasks that are not t nor t_i, and so on. For simplicity, we assume that W_0 and all workflows in rules have nodes labeled by distinct task names. We can then easily prove the following property:

Proposition 5. *Let $C = (W, wa, Dass)$ be a configuration of a recursion free complex workflow CW. Then there exists a bound $K_{\mathcal{T}_{cx}}$ on the size of W, and the length of a (symbolic) execution of CW is in $O(3.K_{\mathcal{T}_{cx}})$*

Proof. [Sketch] We assume, without loss of generality, that all workflows in all rules have nodes labeled by distinct task names, and the initial workflow has a single node. Let d be the maximal number of new occurrences of complex tasks that can be rewritten in one refinement (i.e., the maximal number of complex tasks that appear in a rule). Each rewriting adds at most $d - 1$ complex tasks to the current configuration. The number of rewriting is bounded, as CW is recursion free. For a given node n appearing in a configuration C_k along a run, one can trace the sequence of rewriting $Past(n)$ performed to produce n. According to recursion freeness, when a node n is replaced by a workflow W_t, then none of the tasks labeling nodes of W_t appears in $Past(n)$. Hence, the number of nodes in a configuration is at most $K_{\mathcal{T}_{cx}} = d^{|\mathcal{T}_{cx}|}$. Now, for a given configuration, the number of applications of semantic rules $R1$, $R2$, and $R3$ is bounded and decreases the number of nodes in the workflow part of the configuration. ■

Till now, we showed in proposition 4 that recursion-freeness is decidable and in $O(|\mathcal{T}_{cx}^2|+|\mathcal{R}|)$. Further, letting d denote the maximal number of complex tasks appearing in a rule, there exists a bound $K_{\mathcal{T}_{cx}} \leq d^{|\mathcal{T}_{cx}|}$ on the size of W in a configuration, and the length of a (symbolic) execution of CW is in $O(3.K_{\mathcal{T}_{cx}})$ (see Prop. 5). However, existence of a bound on the size of W in a configuration does not guarantee universal termination. There may exist a run with an empty dataset that can lead to the non-universal termination. We give the following lemma.

Lemma 2. *Let $C_i^S = (W_i, wa_i, Dass_i^S)$ be a potential deadlock with successors $C_{i,1}^S, \ldots C_{i,k}^S$ corresponding respectively to execution of tasks attached to minimal nodes $n_1, \ldots, n_q$ in the workflow part of node W_i. Then a run ρ with signature $\rho^S = C_0^S \ldots C_i^S$ such that $D_k = \emptyset$ for some $D_k \in Dass(n_j), j \in 1, \ldots, q$ does not terminate.*

Proof. If a node n_j in a configuration C_i with signature C_i^S is labeled by a task and is attached to an empty dataset, then any sequence of worker assignment, refinement, or task execution occurring from C_i will result in a configuration C_i' where either n_j is still attached an empty dataset, or n_j was replaced, but the refinement produced fresh nodes with an empty dataset attached to it. Then either n_j or one of its refinements will never be executed, and the workflow cannot reach a final configuration. ■

The lemma has useful consequences: it is sufficient to detect a run with signature $C_0^S \dots C_i^S$ as the prefix, where $C_i^S = (W_i, wa_i, Dass_i^S)$, and to prove that a node n_j in W_i can have an empty input dataset D_{n_j} to claim that there exists an execution that deadlocks in CW. Now, based on the above results, we can now prove that a complex workflow that is not recursion-free does not terminate universally and give the following proposition.

Proposition 6. *A complex workflow terminates universally if and only if:*

(i). it is recursion free

(ii). it has no (symbolic) deadlocked execution

(iii). there exists no run with signature $C_0^S \dots C_i^S$ where C_i^S is a potential deadlock, with $D_k = \emptyset$ for some $D_k \in Dass(n_j)$ and for some minimal node n_j of W_i.

Proof. Let us first prove that if $i)$ fails, a complex workflow does not terminate universally. If CW has recursive task rewriting, then there is a cycle in the rewriting graph $RG(CW)$ that is accessible from a task $t_0 = \lambda(n_0)$ appearing in W_0. Hence, there is an infinite run $\rho^\infty = C_0 \xrightarrow{a_1} C_1 \xrightarrow{r_1} C_2 \dots$ of CW which moves are only worker assignments (a moves) to a node of the current workflow at configuration C_i followed by a rewriting (r moves) that creates new instances of tasks, such that the sequence of rewritten task follows the same order as in the cycle of $RG(CW)$. Similarly, if CW terminates universally, then all runs are finite, and infinite runs of the form ρ^∞ cannot exist, and CW must be recursion free.

We can now address point $ii)$ If CW can reach a deadlocked configuration, then by definition, it does not terminate. If all runs of CW terminate, then from any configuration, there is a way to reach a final configuration, and hence no deadlock is reachable. The last point $iii)$ is proved in lemma 2. ■

Now, we have the decidability results for the universal termination for a complex workflow. Next, we give the complexity analysis. We do not detail the proofs of Theorem 2 and refer to Appendix A.2 for detailed proofs.

Theorem 2. *Let CW be a complex workflow, in which tasks do not use SQL difference. Let D_{in} be an input dataset, and $\mathcal{D}_{in}$ be a FO formula. Universal termination of CW on input D_{in} is in $co-2EXPTIME$. Universal termination on inputs that satisfy $\mathcal{D}_{in}$ is undecidable in general. It is in*

— *$co-2EXPTIME$ (in K, the length of runs) if $\mathcal{D}_{in}$ is in $\forall$FO*

— *$co-3EXPTIME$ if $\mathcal{D}_{in}$ is $\exists$FO or BSR-FO*

— *$co-n_{in}$-fold-$EXPTIME$, where $n_{in} = |\mathcal{D}_{in}| + 2^K$ if $\mathcal{D}_{in}$ is in SF-FO.*

4.2.1 Symbolic Execution Tree

One can notice that the algorithm to check universal termination of CWs guesses a path, and is hence non-deterministic. In the worst case, one may have to explore *all* symbolic executions of size at most $3 \cdot K_{\mathcal{T}_{cx}}$. All these executions can be grouped in a common structure, i.e. a symbolic execution tree representing possible sequences of moves starting from the initial configuration.

Definition 30 (Symbolic Execution Tree)**.** *The* Symbolic execution tree *(SET) of a complex workflow $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ is a pair $\mathcal{B} = (V, E)$, where $E \subseteq V \times V$ is a set of edges, V is a set of symbolic configurations of the form $C_i^S = (W_i, wa_i, Dass^S)$, where $W_i = (N_i, \longrightarrow_i, \lambda_i)$ and $wa_i : N_i \to \mathcal{U}$ are the usual workflow and worker assignment relations, and $Dass^S$ associates a sequence of relational schemas to minimal nodes of W_i. $(C_i^S, C_j^S) \in E$ if $C_j^S \in succ(C_i^S)$.*

Each path of the tree defines a symbolic execution. The symbolic execution tree of a complex workflow is a priori an infinite structure, but for recursion free CWs, the tree is of bounded depth, and bounded degree. One can hence perform an exhaustive search for deadlocks and potential deadlocks in the symbolic execution tree of a recursion free workflow to exhibit a witness for non-termination. Let $C_i^S = (W_i, wa_i, Dass_i)$ be a symbolic configuration that is a potential deadlock. Let $S = \{n_1, \ldots, n_k\} \subseteq min(W_i)$ be the set of minimal nodes that represent data transformations with rule R2 or R3, i.e. that is atomic task completion or represents an automated task completion. Realization of these tasks ask for non-empty inputs. Let $\Pi = C_0^S \ldots C_i^S$ be the path from the root of the tree to a potential deadlock C_i^S. Even if vertex C_i^S has a successor C_k^S, obtained by executing a task attached to some node $n_j \in S$, it can be the case that $Dass_i$ assigns an empty input to n_j in an actual run of CW with signature Π. Hence, some of the task execution move depicted in $\mathcal{B}$ may not be realizable. If executing task $\lambda(n_j)$ is

the only possible action from C_i^S and if a run with signature Π ends in a configuration where $Dass_i(n_j)$ is of the form $D_1 \dots \emptyset \dots D_q$ then the run is deadlocked. However, if all runs with signature Π end with data assignments that affect non-empty sequences of datasets to all nodes of S, then C_i^S will never cause a real deadlock. Note also that when C_i^S is a potential deadlock, there exists necessarily a path $C_i^S.C_{i+1}^S \dots C_{i+h}^S$ in $\mathcal{B}$ where the only actions allowed from C_{i+h}^S is the execution of the incriminated minimal tasks that need non-empty inputs.

4.2.2 Termination with a guaranteed bound

Undecidability of existential termination has several consequences: As complex workflows are Turing complete, automatic verification of properties such as reachability, coverability, boundedness of datasets, or more involved properties written in a dedicated logic such as LTL FO [DDV12] (a logic that addresses both properties of data and runs) are also undecidable. However, one can notice that in the counter machine encoding of Theorem 1 requires refinements and recursive schemes. So, infinite runs of a counter machine can be encoded only if rule 4 can be applied an infinite number of times. Obviously, recursion-free CWs cannot encode counter machines, and have a bounded number of signatures, as their runs are of length at most $3.K_{\mathcal{T}_{cx}}$. This immediately gives us the following corollary:

Corollary 2. *Let CW be a recursion-free complex workflow that does not use dataset difference, which runs length is bounded by $3.K_{\mathcal{T}_{cx}}$. Let D_{in} be a dataset, and $\mathcal{D}_{in}$ a FO formula. One can decide in $2 - EXPTIME$ (in $K_{\mathcal{T}_{cx}}$) whether CW terminates existentially on input D_{in}. If $\mathcal{D}_{in}$ is in the existential, universal or BSR fragment of FO, then existential termination of CW is also in $2EXPTIME$.*

Proof. The proof follows the same line as for Theorem 2: one has to find non-deterministically a signature of size at most $3.K_{\mathcal{T}_{cx}}$. When such a signature ρ^S is found it remains to compute $|\rho^S|$ weakest preconditions $\psi_{|\rho^S|}, \dots \psi_0$, imposing at each step that inputs of automated or split tasks are not empty. After this verification, it remains to show that inputs satisfy ψ_0. ■

Existential termination is hence decidable for recursion-free CWs, provided the description of inputs belongs to a decidable fragment of FO and data transformations do not use difference, which could introduce quantifiers alteration and hence create weakest preconditions that leave this decidable fragment. One can notice that the decision

procedure is doubly exponential in the length of runs (that is in $K_{\mathcal{T}_{cx}}$). This bound can itself be exponential (see proof of prop. 2), but in practice, one can expect refinements to stop only in a few steps, as refinements are supposed to transform a complex task into an orchestration of simpler subtasks. With this assumption, $K_{\mathcal{T}_{cx}}$ remains a simple polynomial in the number of complex tasks. Another way to bound recursiveness in a complex workflow is to limit the number of refinements that can occur during execution. We can slightly adapt the semantics of Section 3.4, and in particular rule R_4, and replace it by a *restrictive decomposition* (RD) rule. Intuitively, the (RD) rule refines a task as in rule R_4 but forbids decomposing the same task an unbounded number of times.

Rule 4' (RESTRICTED TASK REFINEMENT): Let $T = \{t_{int}, t_2, ..., t_f\}$ be a set of tasks of size n. Let $KD = (k_1, k_2, ..., k_n) \in \mathbb{N}^n$ be a vector constraining the number of refinements of task t_i that can occur in a run ρ. In the context of crowdsourcing, this seems a reasonable restriction. Restrictive decomposition RD is an adaptation of rule $R4$ that fixes an upper bound k_i on the number of decomposition operations that can be applied for each task t_i in a run. We augment configurations with a vector $S \in \mathbb{N}^n$, such that $S[i]$ memorizes the number of decomposition of task t_i that have occurred since the beginning of the execution. Rules 1-3 leave counter values unchanged, and rule 4 becomes:

$$\frac{\begin{array}{l} t = \lambda(n) \in \mathcal{T}_{cx} \wedge S[i] \leq k_i \\ \wedge \exists u, u = wa(n) \wedge (u, r_i) \in sk \wedge r_i = (t, W_S) \\ \wedge Dass'(min(W_s)) = Dass(n) \\ \wedge \forall x \in N_s \setminus min(W_s), Dass'(x) = \emptyset^{|Pred(x)|} \\ \wedge wa' = wa \backslash \{(n, wa(n))\} \\ \wedge \forall j \in 1 \ldots |T|, S'[j] = S[j] + 1 \text{ if } j = i, S[j] \text{ otherwise} \end{array}}{(W, wa, Dass) \xrightarrow{ref(n)} (W_{[n/W_s]}, wa', Dass')} \tag{4.1}$$

Following the RD semantics, each task t_i can be decomposed at most k_i times. To simplify notations, we choose a uniform bound $k \in \mathbb{N}$ for all tasks, i.e. $\forall i \in 1..n$, $k_i = k$. However, all results established below extend to a non-uniform setting. We next show the decidability of existential termination under the RD semantics. First, we give an upper bound on the length of runs under RD semantics. Let k be a uniform bound on the number of decompositions, $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ be a complex workflow with

a set of tasks of size n, and $C_0 = (W_0, wa_0, Dass_0)$ be its initial configuration. Hence, RD semantics bounds the number of tasks in the workflow and we get the following propositions.

Proposition 7. *Let $\rho = C_0.C_1 \dots C_q$ be a run under RD semantics. Then, for every $C_i = (W_i, wa_i, Dass_i)$, the number of nodes in W_i is smaller than $NmaxNodes(n,k) = k.n^2 + |W_0|$.*

Proof. Each decomposition of a task t_i replaces a single node n by a new workflow with at most $d_i = \max_{u \in \mathcal{U}} \max\{|W_j| \mid W_j \in Profile(t_i, u)\}$ nodes. Recall that decomposition profiles are known and that all nodes of workflows in profiles are attached to distinct task names. So, we have $d_i < n$. Every run ρ starting from C_0 is a sequence of rule applications. Rule 1 does not affect the size of workflows in configurations, and rules 2 and 3 remove at most one node from the current workflow when applied. For each task t_i, a run ρ contains at most k occurrences of rule 4 refining a task of type t_i. Application of rule 4 to task t_i adds at most d_i nodes to the current workflow and removes the refined node. All other rules decrease the number of nodes. One can notice that each task can be decomposed at most k times, rule 4 can be applied at most $k.n$ times in a run following the RD semantics, even if this run is of length greater than $k.n$. Let $S_0 = |W_0|$, $S_1 = S_0 + n - 1$, and $S_{i+1} = S_i + (n-1)$. For a fixed n and a fixed k, the maximal size of the workflow component W_i in every configuration C_i of a run under RD semantics is smaller than $S_{k.n} = |W_0| + (k.n)(n-1) = |W_0| + k \cdot n^2 - k \cdot n$. ■

For example, if the number of tasks is $n = 4$, the size of the initial workflow 2, and the number of decomposition uniformly bounded by $k = 3$, then $NmaxNodes(n,k) = 38$. Now, as the number of tasks is bounded during the realization of a complex workflow, it gives us the bound on the length of the run.

Proposition 8. *Let $\rho = C_0 \dots C_q$ be a run of a complex workflow under RD semantics allowing at most k refinements of each task. The length of ρ is bounded by $L(n,k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$*

Proof. Recall that a configuration is a triple $C_i = (W_i, wa_i, Dass_i)$. Each configuration is a "global state" of the execution of a complex workflow. W_i represents the work that needs to be executed, wa the worker's assignment, and $Dass$ the data assignment. Recall that a configuration with a single node is necessarily a final configuration with a node n_f which task is to return all computed values during the execution of the complex workflow.

The only way to change the data assignment part of configurations is to execute the task attached to a node (i.e., apply rule $R2$ or $R3$) or refine a node (i.e. apply $R4$). Starting from a configuration C_i, the maximal number of worker assignment that can be performed is $|W_i|$, and along the whole run, as each node can be assigned a worker at most once, the maximal number of applications of rule $R1$ is $NmaxNodes(n,k)$.

The length of a run ρ is $|\rho| = |\rho|_1 + |\rho|_2 + |\rho|_3 + |\rho|_4$ where $|\rho|_i$ denotes the number of applications of rule R_i. Now, $|\rho|_1 \leq NmaxNodes(n,k)$. Similarly, $|\rho|_1 = |\rho|_2 + |\rho|_4$. Last, rule $R3$ can be applied only a number of times bounded by the maximum number of created nodes, i.e, $|\rho|_3 \leq NmaxNodes(n,k)$. So overall, $|\rho| = |\rho|_1 + (|\rho|_2 + |\rho|_4) + |\rho|_3 \leq NmaxNodes(n,k) + NmaxNodes(n,k) + NmaxNodes(n,k)$. Hence, the length of ρ is bounded by $L(n,k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$ ■

Note that here, configurations can only grow up to a size smaller than $NmaxNodes(n,k) = k.n^2 + |W_0|$ (Prop. 7) via rule R4, and rules R1-R3 can be applied only a finite number of times from each configuration. Under RD semantics, a symbolic execution tree $\mathcal{B}$ is necessarily finite and of bounded depth. A run terminates iff it goes from the initial configuration to a final one. If such run exists, then there exists a path in $\mathcal{B}$ from the initial vertex to a final vertex with signature $\Pi = V_0 \dots V_n$. Further, if this path visits a potential deadlock and executes a splitting task $\lambda(n)$ for some split node n_j, then every dataset used as input of n_j must be non-empty. To show that this path is realizable, it suffices to show the existence of a run with signature Π that ends in a configuration C_n satisfying property $\phi ::= true$. Proposition 2 shows how to compute backward the weakest preconditions demonstrating the existence of such run of CW. An immediate consequence is that the existential termination of complex workflows is decidable under restricted decomposition semantics, with the same complexity as for recursion-free specifications.

Theorem 3. *Let CW be a complex workflow in which tasks do not use SQL difference, and which runs are of length $\leq K_{\max}$. Let D_{in} be a dataset, and $\mathcal{D}_{in}$ a FO formula. One can decide in $2-EXPTIME$ (in $K_{\max}$) whether CW terminates existentially on input D_{in}. If $\mathcal{D}_{in}$ is in $\exists$FO, termination of CW is also in $2-EXPTIME$. It is in $3-EXPTIME$ when $\mathcal{D}_{in}$ is in $\forall$FO or BSR-FO*

Proof. We can reuse the techniques of Theorem 2 to find a witness symbolic run that is the signature of a run that does not terminate, and of corollary 2 to find a witness symbolic run that is the signature of a run that terminates. ■

4.3 Correctness of Complex Workflows

Complex workflows provide a service to a client, that inputs some data (a dataset D_{in}) to a complex task, and expects some answer, returned as a dataset D_{out}. A positive answer to a termination question means that the process specified by a complex workflow does not deadlock in some/all executions. A client sees the crowdsourcing platform as a black box and simply asks for the realization of a complex task that needs specific competencies. Even when termination is guaranteed, the returned data can still be incorrect. In many cases, a client may have requirements on the type of output returned for a particular input. We express this constraint with a FO formula $\psi_{in,out}$ relating inputs and outputs and extend the notions of existential and universal termination to capture the fact that a complex workflow implements client's needs if some/all runs terminate, and in addition fulfill requirements $\psi_{in,out}$. This is called *correctness*.

Definition 31. *A* constraint *on inputs and outputs is a FO formula*

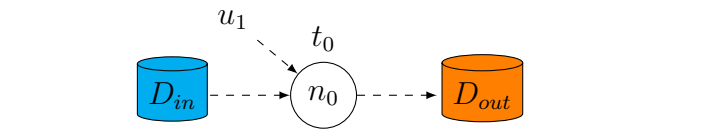
$$\psi_{in,out} ::= \psi^{in,out_E} \wedge \psi^{in,out_A} \wedge \psi^{in,out_{AE}} \wedge \psi^{in,out_{EA}}\text{, where}$$

- ψ^{in,out_E} *is a conjunction of* $\exists$*FO formulas addressing the contents of the input/output dataset, of the from* $\exists x, y, z, rn(x, y, z) \in D_{in} \wedge P(x, y, z)$ *or* $\exists u, v, w, rn(u, v, w) \in D_{out} \wedge P(u, v, w)$*, where* $P(.)$ *is a predicate.*
- ψ^{in,out_A} *is a conjunction of* $\forall$*FO formulas constraining all tuples of the input/output dataset, of the form* $\forall x, y, z, rn(x, y, z) \in D_{in} \Rightarrow P(x, y, z)$.
- $\psi^{in,out_{AE}}$ *is a conjunction of formulas relating the contents of inputs and outputs, of the form* $\forall x, y, z, rn(x, y, z) \in D_{in} \Rightarrow \exists(u, v, t), \varphi(x, y, z, u, v, t)$*, where* φ *is a predicate.*
- $\psi^{in,out_{EA}}$ *is a conjunction of formulas relating the contents of inputs and outputs, of the form* $\exists x, y, z, rn(x, y, z) \in D_{in}, \forall(u, v, t), \varphi(x, y, z, u, v, t)$.

The $\psi^{in,out_{AE}}$ part of the I/O constraint can be used to require that every record in an input dataset is tagged in the output. An example is shown in Figure 4.4. Consider a node n_0 tagged with task $\lambda(n_0) = t_0$ and the attached input dataset $movie(Id, title, year) \in D_{in}$. The objective of task t_0 is to tag every record in the dataset D_{in} such that $rating \geq 4$. In this case, $\psi^{in,out_{AE}}$ constraint is

$$\forall id, t, y, r\ movie(id, t, y, r) \in D_{in} \implies \exists id, t, y, r\ movie(id, t, y, r) \in D_{out} \wedge (r \geq 4)$$

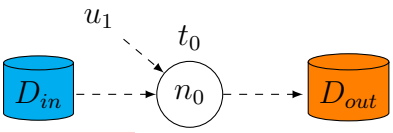
The ψ_{EA} part can be used to specify that the output is a particular record selected from the input dataset (to require correctness of a workflow that implements a vote).



Id	$title$	$year$
1	.	.
2	.	.
3	.	.
4	.	.

Id	$title$	$year$	$rating$
1	.	.	5
2	.	.	8
3	.	.	6
4	.	.	8

Figure 4.4 – An example showing ψ_{AE} constraints.



Id	$title$	$year$	$rating$
1	Star Wars	.	4
2	Titanic	.	5
3	Star Wars	.	4
4	Titanic	.	5
4	Titanic	.	5

Id	$title$	$year$	$rating$
2	Titanic	.	5
4	Titanic	.	5
4	Titanic	.	5

Figure 4.5 – An example showing ψ_{EA} constraints.

Figure 4.5 shows an example of ψ_{EA} constraints. Consider a node n_0 with input dataset $movie(Id, title, year) \in D_{in}$. The node n_0 is tagged with task t_0 with an objective to select a *movie* such that each *rating* of the *movie* is 5. In this case, ψ_{EA} constraint is given as

$$\exists id, t, y, r\ movie(id, t, y, r) \in D_{in} \forall r\ movie'(id, t, y, r) \in D_{out} \implies (r = 5)$$

Definition 32 (Correctness)**.** *Let CW be a complex workflow, $\mathcal{D}_{in}$ be a set of input datasets, and $\psi_{in,out}$ be a constraint given by a client. A run in $\mathcal{R}uns(CW, D_{in})$ is* correct *if it ends in a final configuration and returns a dataset D_{out} such that $D_{in}, D_{out} \models \psi_{in,out}$. CW is existentially correct with inputs $\mathcal{D}_{in}$ iff there exists a correct run $\mathcal{R}uns(CW, D_{in})$ for some $D_{in} \in \mathcal{D}_{in}$. CW is universally correct with inputs $\mathcal{D}_{in}$ iff all runs in $\mathcal{R}uns(CW, D_{in})$ are correct for every $D_{in} \in \mathcal{D}_{in}$.*

In general, termination does not guarantee correctness. A terminated run starting from an input dataset D_{in} may return a dataset D_{out} such that pair D_{in}, D_{out} does not

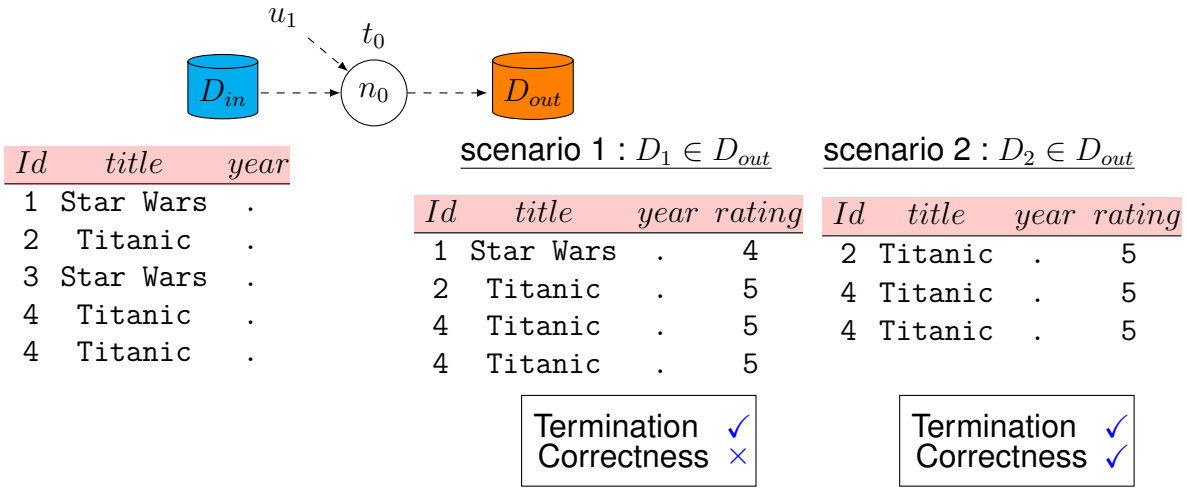


Figure 4.6 – Different scenario of a terminated run: with and without correct set of output.

comply with constraint $\psi_{in,out}$ imposed by the client. For instance, a run may terminate with an empty dataset while the client required at least one answer. Similarly, a client may ask all records in the input dataset to appear with an additional tag in the output. If any input record is missing, the output will be considered incorrect. Figure 4.6 depicts two scenarios. Consider the node n_0 tagged with task t_0 is attached with dataset $movie(Id, title, year) \in D_{in}$. Client requirement is to get a dataset D_{out} with *movie* rating greater than or equal to 5.

$$\forall id, t, y, r\ movie(id, t, y, r) \in D_{in} \implies \exists id, t, y, r\ movie(id, t, y, r) \in D_{out} \wedge (r \geq 5)$$

Here in Figure 4.6, we show two scenarios. In the first case, task t_0 terminates, but with the wrong set of output $D_1 \in D_{out}$ which was not desired by the client, i.e. the record $movie(1, Star\ Wars, ., 4)$ do not meets the client requirement, $movie(1, Star\ Wars, ., 4) \not\models \psi_{in,out}$. On the other hand, in the second scenario, the task t_0 terminates with the correct output $D_2 \in D_{out}$, and satisfies client requirement $D_2 \models \psi_{in,out}$. Here, we can observe that termination does not guarantee the correctness and hence requires mechanisms to guarantee correctness of complex workflow. As for termination, correctness can be handled through symbolic manipulation of datasets but has to consider constraints that go beyond the emptiness of datasets. Weakest preconditions can be effectively computed (Prop. 1): one derives successive formulas $\psi_i^{in,out}, \ldots \psi_0^{in,out}$ between D_{in}, D_{out} and datasets in use at step $i, \ldots 0$ of a run. However, the $\psi_{AE}^{in,out}$ part

of formulas is already in an undecidable fragment of FO, so even universal termination is undecidable in general, and even when a bound on the length of runs is known. It becomes decidable only with some restrictions on the fragment of FO used to write $\psi^{in,out}$.

Theorem 4. *Existential and universal correctness of CW are undecidable, even when runs are of bounded length K. If tasks do not use SQL difference, and $\psi_{in,out}$ is in a decidable fragment of FO, then*

- *existential correctness is decidable for CWs with runs of bounded length. it is in*
  - $2EXPTIME$ *for the $\exists$FO.*
  - $3EXPTIME$ *for $\forall$FO, BSR.*
  - 2^K*-fold-*$EXPTIME$ *for SF fragments.*
- *universal correctness is decidable for CWs with runs of bounded length. It is in*
  - $co-2EXPTIME$ *for the $\exists$FO.*
  - $co-3EXPTIME$ *for $\forall$FO, BSR.*
  - co-2^K*-fold-*$EXPTIME$ *for SF fragments.*

Proof. Let us first prove the undecidability part: It is well known that satisfiability of FO is undecidable in general, and in particular for the AE fragment with formulas of the form $\forall\vec{X}\exists\vec{Y}, \phi(X,Y)$. Hence $\psi_{in,out_{AE}}$ can be a formula in which satisfiability is not decidable. Consider for example a FO formula ψ_{unsat} which satisfiability is not decidable. One can then build a formula ψ_{id} that says that the input and output of a workflow are the same. One can design a workflow CW_{id} with a single final node which role is to return the input data, and set as client constraint $\psi_{in,out} = \psi_{unsat} \wedge \psi_{id}$. This workflow has a single run. Then, CW_{id} terminates properly if there exists a dataset D_{in} such that $D_{in} \models \psi_{unsat}$, i.e. if ψ_{unsat} is satisfiable. Universal and existential correctness are hence undecidable problems.

For the decidable cases, one can apply the technique of Theorem 2. One can find non-deterministically a symbolic run ρ^S that does not terminate and check that it is the signature of an actual run, or a symbolic run ρ^S that terminates and check whether it satisfies $\psi_{in,out}$.

Let us first consider universal correctness. Assume that CW terminates universally, and select a symbolic run $\rho^S = C_0^S \dots C_n^S$. We can then compute a chain of weakest preconditions $\psi_n, \psi_{n-1}, \dots \psi_0$ that have to be enforced to execute successfully CW and terminate in node n. In particular, $\psi_n ::= true$. Similarly, one can compute at each step, the weakest precondition ψ_{in,out_i} needed at step i so that $\psi_{in,out}$ holds. Intuitively,

ψ_{in,out_i} describes the constraints between the initial dataset and the output dataset "consumed" at stage $i+1$ in ρ^S. If at one stage, $\psi_i \wedge \psi_{in,out_i}$ is not satisfiable, then ρ^S is not the signature of an actual run of CW that terminate properly, and we have found a witness of non-correctness. We have assumed that $\psi_{in,out}$ was specified in a decidable fragment of FO. As computing the weakest precondition of a property in the existential, universal, BSR, SF fragment of FO gives property in the same fragment, all ψ_i's and ψ_{in,out_i}'s are in a decidable fragment of FO. Then, the complexity will depend on the considered fragment, and on the fragment of FO used to specify inputs. As for universal termination, if inputs and $\psi_{in,out}$ are specified with the universal fragment of FO, then universal proper termination is in $co-2EXPTIME$, and in $co-3EXPTIME$ for the existential fragment (as one may alternate $\exists$ statements on outputs with $\forall$ statements inherited from the obligation to prove non-emptiness of a dataset. Similar remark and complexity hold for the BSR fragment (separation of variables maintains a NEXPTIME complexity [SVW16]). If $\psi_{in,out}$ is in SF, then checking proper universal termination is $co-K-fold-$exponential time, where $K = r^{K_{\mathcal{T}_{cx}}}$.

The proof and complexities for existential correctness follow the same lines, yielding $2EXPTIME$ complexity when $\psi_{in,out}$ is written with the existential, fragment of FO, $3-EXPTIME$ complexity for when $\psi_{in,out}$ is written in the universal or BSR fragments (as checking satisfiability for a BSR formula is in NEXPTIME [Lew80]) and $K-fold-$exponential for SF formulas [SVW16]. ■

At first sight, restricting to the existential, universal, BSR, or SF fragments of FO can be seen as a limitation. However, the existential fragment of FO is already a very useful logic, that can express non-emptiness of outputs: property $\exists x_1, \ldots, \exists x_k, rn(x_1, \ldots x_k) \in D_{out}$ expresses the fact that the output should contain at least one record. Similarly, one can express properties to impose that every input has been processed. For instance, the property

$$\psi^{valid}_{in,out} ::= \forall x_1 \ldots x_k, rn(x_1, \ldots x_k) \in D_{in}\ \exists y_1 \ldots y_q, rn(x_1, \ldots x_k, y_1, \ldots y_q) \in D_{out} \\ \wedge P(x_1, \ldots x_k, y_1, \ldots y_q)$$

asks that every input in D_{in} appears in the output, and $P()$ describes correct outputs. Clearly, $\psi^{valid}_{in,out}$ is not in the separated fragment of FO. This formula can be rewritten into another formula (BSR form) with a single alternation of quantifiers of the form:

$$\forall x_1 \ldots x_k, \exists y_1 \ldots y_q, \neg rn(x_1, \ldots x_k) \in D_{in} \\ \vee\ rn(x_1 \ldots x_k, y_1 \ldots y_q) \in D_{out}$$

Workflow Type	**FO Fragment (for $\mathcal{D}_{in}$ or $\psi^{in,out}$)**	**Problems Complexity (no SQL diff.)**	
		Existential Termination	**Universal Termination**
Static, Recursive Bounded	FO	Undecidable	Undecidable
	$\exists^*(\forall^*$ if univ. PB$)$	$2EXPT$	$co-2EXPT$
	BSR,$\forall^*(\exists^*$ if univ. PB$)$	$3EXPT$	$co-3EXPT$
	SF	$n_{in}-foldEXPT$	$co-n_{in}$-fold-$EXPT$
Recursive Unbounded	FO	Undecidable	Undecidable
	$\exists^*(\forall^*$ if univ. PB$)$	Undecidable	$co-2EXPT$
	BSR,$\forall^*(\exists^*$ if univ. PB$)$	Undecidable	$co-3EXPT(K)$
	SF	Undecidable	$co-n_{in}-foldEXPT$

Table 4.1 – Complexity of Termination ($EXPT$ stands for EXPTIME).

Workflow Type	**FO Fragment (for $\mathcal{D}_{in}$ or $\psi^{in,out}$)**	**Problems Complexity (no SQL diff.)**	
		Existential Correctness	**Universal Correctness**
Static, Recursive Bounded	FO	Undecidable	Undecidable
	$\exists^*(\forall^*$ if univ. PB$)$	$2EXPT$	$co-2EXP$
	BSR,$\forall^*(\exists^*$ if univ. PB$)$	$3EXPT$	$co-3EXPT$
	SF	$2^{K\tau_{cx}}$-fold-$EXPT$	co-$2^{K\tau_{cx}}$-fold-$EXPT$
Recursive Unbounded	FO	Undecidable	Undecidable
	$\exists^*(\forall^*$ if univ. PB$)$	Undecidable	$co-2EXPT$
	BSR,$\forall^*(\exists^*$ if univ. PB$)$	Undecidable	$co-3EXPT$
	SF	Undecidable	co-$2^{K\tau_{cx}}$-fold-$EXPT$

Table 4.2 – Complexity of Correctness ($EXPT$ stands for EXPTIME).

Last, one can also consider formulas in which $\psi^{in,out_{EA}}$ is of the form $\forall x_1, \ldots x_k \exists y_1, \ldots, y_q\ \phi$ as soon as every atom in ϕ that is not separated contains only existential variables that take values from a finite domain. Then $\psi^{in,out_{EA}}$ can be transformed into an equivalent universal formula which matrix is a boolean expression on separated atoms.

Table 4.1summarizes the complexities of termination and Table 4.2 summarized the complexities of correctness for static complex workflows (without higher-order answer) or with bounded recursion, and for generic workflows with higher order.

4.4 Use Case

We illustrate the algorithm to check termination on a simple use case, namely an image annotation task. The scenario is the following: a client wants to realize a complex task whose global objective is to annotate images. The objective is to tag a huge dataset of images with a predefined taxonomy. The initial workflow consists of three nodes n_{int}, n_1 and n_f and is depicted in Figure 4.7(left). The node n_{int} is the initial node and n_f is the final node. The objective of node n_f is to return the final answer to the client. Consider the node n_1 tagged with task t_1 is allocated a worker u_1 for the task execution. The node n_1 receives as input dataset D_1 with relational schema $picdata(id, image)$. The dataset D_1 contains 1000 record with id ranging from 0 to 999 in $picdata$ table. The field $image$ is an image object URL (to retrieve images). The objective of task t_1 is to tag each of the records from a predefined taxonomy as Bee or Fly. Considering the huge work that cannot be realized by a single worker, u_1 decides to rewrite the task into another workflow W' by applying the rule R_4. The new workflow W' is represented in Figure 4.7(right). The idea used by the worker u_1 is simple. He decides to create two subsets of the dataset that consists of 500 images each.

The node n_2 performs a *decomposition* of dataset. The input to the node n_2 is dataset D_2 with relational schema $picdata(id, image)$. A *selection* operation is performed to produce a dataset D_3 with relational schema $picdata(id, image)$ with records id smaller than 500 and another *selection* operation that selects records from 500 to 999 to produce dataset D_4 with relational schema $picdata(id, image)$. The dataset D_3 and D_4 acts as an input to the node n_3 and n_4 respectively. The task $t_3 \in n_3$ tags each record from D_3 with a tag from the pre-defined set of $taxon \in \{Bee, Fly\}$ and produces dataset D_5 with relational schema $picdata(id, image, taxon)$. Similarly the node n_4 annotates each of the images and inserts a $taxon \in \{Bee, Fly\}$ in the $taxon$ field and returns $picdata(id, image, taxon) \in D_6$. The outputs of node n_3 and n_4 are forwarded to node n_5. The task t_5 attached to node n_5 performs a *union* operation and produces dataset D_7 with relational schema $picdata(id, image, taxon)$. The node n_6 gets dataset D_7 as input and does a *projection* on the fields $id, taxon$. It returns a datset D_8 with relational schema $picdata(id, taxon)$ which is forwarded to node n_f. The node n_f simply returns the output to the client.

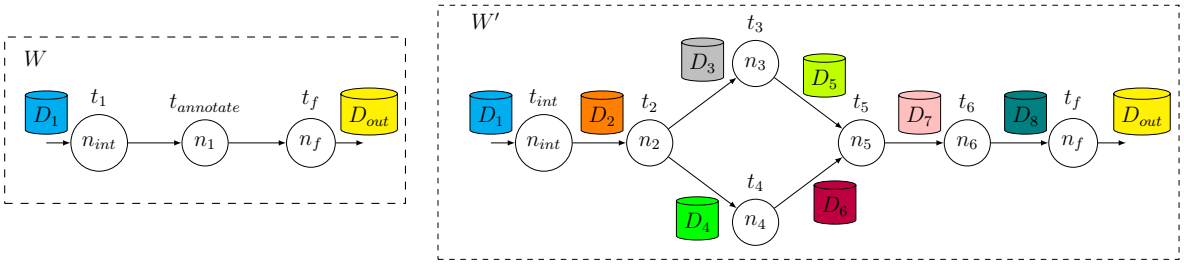


Figure 4.7 – Original workflow W(left). The new workflow is represented as W' after refinement of node n_1 in worklow W(right).

Now, we illustrate the universal termination problem and the objective is to check whether the given workflow W universally terminates. Recall that, for each of the rule execution on a complex workflow we get a run $\rho = \{C_0.C_1 \ldots .C_k\}$ where C_i denotes the configuration of the workflow (see def. 22). A complex workflow does not terminate amounts to proving that it is not recursion-free, or that it has a finite symbolic run which preconditions allow a deadlock. To check a workflow is not recursion-free, we first build a rewriting graph $RG(CW)$ and check the existence of a cycle. In our use case, we can observe there exists no cycle as there is no task t_i that is refined infinitely. Here, we do not explain how to check existense of a cycle in a rewriting graph $RG(CW)$, which can be done with standard algorihms using DFS. Next, the universal termination builds on checking all symbolic runs in the workflow.

To consider all runs of the complex workflow, we build its symbolic execution tree (SET) (see def. 30) which represents all the symbolic runs.

4.4.1 Formulation of Symbolic Execution Tree

Even for a small Complex Workflow, Symbolic Execution Tree can be very large. Hence, we do not represent the whole SET of the CW of Figure 4.7, but rather concentrate on interesting and important parts. The SET is shown in Figure 4.8.

C_0^S denotes the initial symbolic configuration. The n_{int} is the source node tagged with task t_1 that forwards the data to the node n_1. The execution gives the next symbolic configuration C_1^S. The node n_1 attached with dataset D_1 rewrites the task t_1 (Rule 4) and gives symbolic configuration C_2^S. The configuration C_3^S is obtained after the

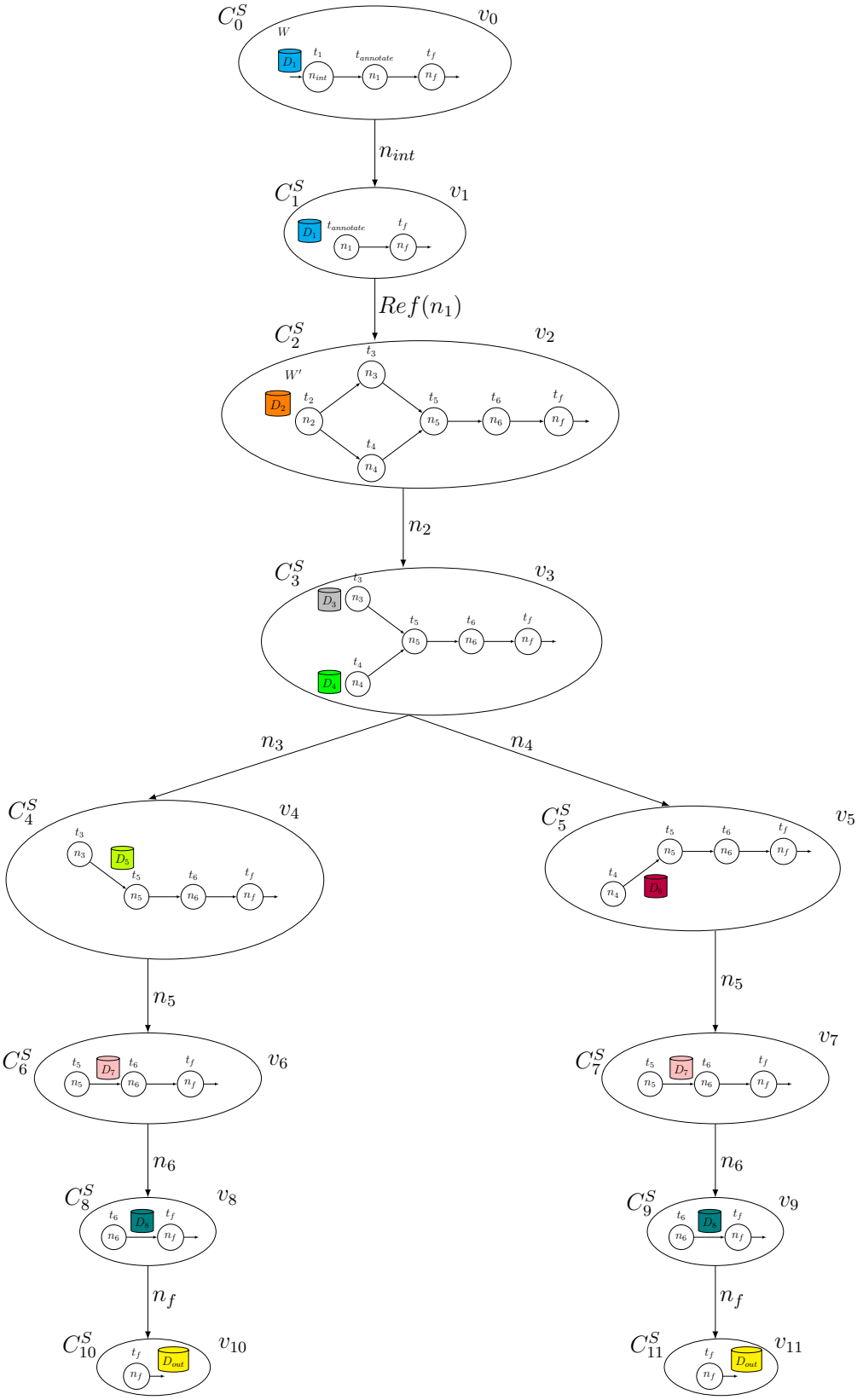


Figure 4.8 – Symbolic Execution Tree.

decomposition of dataset D_2 into dataset D_3 and D_4 by the node n_2. Next we get two symbolic configurations due to decomposition of data D_2. The symbolic configuration C_4^S depicts the execution of an atomic task $t_3 \in n_3$ that performs *insertion* operation on input dataset D_3 and returns dataset D_5. Similarly, The symbolic configuration C_5^S denotes the execution of an atomic task $t_4 \in n_4$ that performs *insertion* operation on input dataset D_4 and returns dataset D_6. From the C_4^S symbolic configuration, we move to C_6^S that depicts the execution of task $t_5 \in n_5$ and performs *union* operation on input dataset D_5, D_6 and returns dataset D_7. Similarly, we move from C_5^S to C_7^S that depicts the *union* operation for node n_5. From the symbolic configurations C_6^S, C_7^S we move to configurations C_8^S and C_9^S respectively. The symbolic configurations C_8^S and C_9^S denote the execution of task $t_6 \in n_6$ that performs *projection* operation on input dataset D_7 and returns output dataset D_8. The symbolic configurations C_9^S and C_{11}^S depict the execution of final node n_f that returns output dataset D_{out}.

4.4.2 Algorithm to check termination

One needs to check all the symbolic runs of the workflow to check universal termination. To recall, for each of the symbolic runs in the SET, the algorithm checks the existence of an empty dataset. We derive the weakest precondition inductively for each of the symbolic configurations and check the satisfiability of the FO formula. If there exist no runs with an empty dataset, the algorithm returns universal termination.

Note that, in the above section, we build a SET that contains all the symbolic runs of the workflow. The leaf nodes of the SET are the symbolic configurations C_{10}^S and C_{11}^S. Both symbolic configurations represent the execution of the final node n_f that returns the final output dataset, hence does not transform data. Similarly the symbolic configurations C_0^S, C_1^S, C_2^S does not transform data. For checking universal termination, we only consider the symbolic configurations that affect data. Hence, we get two symbolic runs as $\rho_1^S = \{C_8^S.C_6^S.C_4^S.C_3^S\}$ and $\rho_2^S = \{C_9^S.C_7^S.C_5^S.C_3^S\}$. Next, to check the workflow universally terminates, for each of the runs we check the satisfiability of the sequence of the weakest precondition from the last symbolic configuration to the initial symbolic configuration in ρ_i^S proving the existence of a run leading to a configuration where a dataset is not empty. Conversely, we check the existence of an empty dataset for each of the runs i.e. in our case ρ_1^S and ρ_2^S. We check the feasibility of ρ_1^S starting from the configuration C_8^S to check all conditions that met along a run with signature to reach C_3^S

at each step that allows the existence of actual configurations. The nodes associated with the symbolic configurations in the run ρ_1^S are n_6, n_5, n_3 and n_2 where operations transformed the datasets. Similarly, the nodes associated with the symbolic configurations in the run ρ_2^S are n_6, n_5, n_4 and n_2 where operations transformed the datasets. Next, we show the calculus to derive the weakest precondition at each of the symbolic configurations and then show inductively checking of weakest preconditions in a symbolic run. We show the steps for the run ρ_1^S.

We start from the symbolic configuration C_8^S obtained after realization of node n_6. The task t_6 at node n_6 takes input dataset D_7, performs a *projection* operation on the fields $id, taxon$ and returns the dataset D_8. The weakest precondition at node n_6 is $wp_{n_6} = wp[Projection]\psi$, where ψ is $D_8 \neq \phi$ which is equivalent to $\exists id, taxon, picdata(id, taxon) \in D_8$. We derive the weakest precondition to check whether the set of operation can lead to a non empty dataset $D_8 \neq \phi$. Here $wp_{n_6} = wp[Projection]\psi$ is a formula

$$\exists v_{id}, v_{taxon}\ picdata(v_{id}, v_{image}, v_{taxon}) \in D_7$$

We trace backward in run ρ_1^S and go to the predecessor symbolic configuration of C_8^S which is C_6^S. The move from C_6^S to C_8^S is the realization of node n_5. The operation at node n_5 is *union* that gets input from node n_3 and n_4. The input dataset provided is $picdata(id, image, taxon) \in D_5$ and $picdata(id, image, taxon) \in D_6$ and produces an output dataset D_7 with relational schema $picdata(id, image, taxon) \in D_7$. Note that, condition for the produced output data $D_7 \neq 0$, is $D_5 \neq 0, D_6 \neq 0$. The weakest precondition at node n_5 is denoted as $wp_{n_5} = wp[Union]\psi$ where $\psi = (D_7 \neq 0)$ and can be expressed with a FO formula as $\exists id, image, taxon, picdata(id, image, taxon) \in D_7$. Hence, wp_{n_5} is a formula

$$\exists u_{id}, u_{image}, u_{taxon}\ picdata(u_{id}, u_{image}u_{taxon}) \in D_6 \vee picdata(u_{id}, u_{image}u_{taxon}) \in D_7$$

The node associated with symbolic configuration C_4^S is n_3 that performs *insertion* operation and inserts $taxon \in \{bee, fly\}$ to the input dataset D_3 with relational schema $picdata(id, iamge, taxon) \in D_3$ and produces dataset D_5 with relational schema $picdata(id, image, taxon) \in D_5$. The weakest precondition at symbolic configuration C_4^S is $wp_{n_3} = wp[Insertion]\psi$, where ψ is $D_5 \neq \phi$ which is equivalent to $\forall id, image, taxon, picdata(id, image, taxon) \in D_5 \wedge taxon \in \{bee, fly\}$. The weakest precondition for the node n_3 is a formula

$wp_{n_3} = wp[Insertion]\psi$ is

$$\exists u_{id}, u_{image}\ picdata(u_{id}, u_{image}) \in D_3$$

Similarly, the node associated with symbolic configuration C_5^S is node n_4 that performs an *insertion* operation and takes input dataset D_4 with relational schema $picdata(v_{id}, v_{image} v_{taxon}) \in D_4$. It tags each record with a $taxon \in \{bee, fly\}$ and produces dataset D_6. The weakest precondition at symbolic configuration C_5^S is $wp_{n_4} = wp[Insertion]\psi$, where ψ is $D_6 \neq \phi$ which is equivalent to $\forall id, image, taxon, picdata(id, image, taxon) \in D_6 \wedge taxon \in \{bee, fly\}$. The weakest precondition for the node n_4 is a formula $wp_{n_4} = wp[Insertion]\psi$ is

$$\exists u_{id}, u_{image}\ picdata(u_{id}, u_{image}) \in D_4$$

At last, we derive the weakest precondition for the symbolic configuration C_3^S which is associated with task t_2 that performs the *decomposition* operation. The *decomposition* operation is a higher order operation and splits the input dataset D_2 into a set of dataset D_3, D_4. The dataset D_3 and D_4 is obtained using $Selection$ operation on the input dataset D_2 with relational schema $picdata(id, image, taxon)$. D_2 is obtained through selection predicates $P_1 :: (id \geq 0 \wedge id \leq 499)$ and $P_2 :: (id \geq 500 \wedge id \leq 999)$. The weakest precondition at node n_2 is $wp_{n_2} = wp[Decomposition]\psi$, where ψ is $D_3 \neq \phi$ and $D_4 \neq \phi$ which is equivalent to $\exists u_{id}, u_{image}, u_{taxon}, v_{id}, v_{image}, v_{taxon}$ $picdata(u_{id}, u_{image}, u_{taxon}) \in D_3 \wedge v_{id}, v_{image}, v_{taxon} \in D_4$. Then weakest precondition for the decomposition operation $wp_{n_2} = wp[Decomposition]\psi$ is the formula

$$\begin{aligned}&\exists u_{id}, u_{image}\ picdata(u_{id}, u_{image}) \in D_1 \wedge (u_{id} \geq 0 \wedge u_{id} \leq 499)\\ &\vee picdata(u_{id}, u_{image}) \in D_2 \wedge (u_{id} \geq 500 \wedge u_{id} \leq 999)\end{aligned}$$

Here, we derived inductively a sequence $wp_{n_6}.wp_{n_5}.wp_{n_3}.wp_{n_2}$ of weakest conditions for the symbolic run $\rho_1^S = \{C_8^S.C_6^S.C_4^S.C_3^S\}$ which is to be met at each stage such that condition $D_i \neq \emptyset$ at a node n_i (hence not leading to an unavoidable deadlock). At each stage, we check the satisfiability of the derived weakest precondition wp_{n_i} using a solver, Z3 [DMB08]. Now, for a derived precondition at a node, if the FO formula is not satisfiable then we return *non-termination*, otherwise, we inductively go back from wp_n to wp_{n-1} and repeat the procedure up to the derivation of the weakest precondition for the initial symbolic configuration of the symbolic run. Following our use case, if

the weakest precondition wp_{n_6} is not satisfied, we return *non-termination*, otherwise we take one step back and derive the weakest precondition wp_{n_5}. We repeat the procedure up to wp_{n_2}. If a run exists such that we are able to reach from the last symbolic configuration C_8^S to the initial symbolic configuration C_3^S in run ρ_1^S i.e. all derived weakest precondition in the path from wp_{n_6} to wp_{n_2} satisfies, then it proves that there does not exist a configuration which leads to an empty dataset. Similarly, analogy follows for the run ρ_2^S where the sequence of derived weakest preconditions are $wp_{n_6}.wp_{n_5}.wp_{n_4}.wp_{n_2}$. Note that, if for all runs, we are able to trace back to the initial symbolic configuration, the algorithm proves universal termination. In our use case, for each of the symbolic run ρ_1^S and ρ_2^S, we find that all the derived weakest precondition along the path is satisfiable (using Z3 solver [Pro]) and hence the algorithm returns verdict as universal termination. The syntactical derivation of the weakest precondition and checking satisfiability using Z3 allows obtaining the simulation and satisfiability results within a reasonable time of a few seconds.

4.5 Platform

We implement the proposed complex workflow model of Chapter 3 and algorithms to check termination and correctness properties. The tool is named **CrowdPlex**.

The tool is developed in JAVA. We design the CrowdPlex tool as a modular system composed of three components: Workflow specification, Operational semantics, and Property checker. The **workflow specification** module consists of several sub-components: coarse description of task in the form of *workflow*, *task mapping*, *profile*, *worker availability*, *worker skills*, *task pre-requisite* and *data* specification. Workflow specification is provided as *text file* or can be passed dynamically at run time to the CrowdPlex tool. Each sub-components of workflow specification follows a predefined grammar and is parsed using a parser developed with JavaCC (Java Compiler Compiler) [1]. The **operational semantics** module gives various rules that are the guiding principles for the execution of a complex workflow. The operational semantics follows the four rules as described in Section 3.4. They describe how a configuration is transformed when a particular action takes place (worker assignment, task completion, ...). The last part of CrowdPlex is a **property checker** module to verify termination and correctness properties. The module generate symbolic runs, synthesizes the weakest

1. **https://javacc.github.io/javacc/**.

preconditions and relies on the Z3 solver to check the satisfiability of a first-order formula. Z3 is an efficient and well-known Satisfiability Modulo Theories (SMT) solver with specialized algorithms and is developed by Microsoft [Pro]. SMT generalizes boolean satisfiability (SAT) problems by adding equality, reasoning, quantifiers, and first-order theories.

Working of CrowdPlex. The first module of the CrowdPlex tool, *workflow specification* takes the input as crowdsourcing attributes and the coarse description of a complex task. The *operational semantics* module interacts with the workflow specification module and applies the various defined rules for the workflow execution. The *property checker* module implements the existential and universal termination algorithm.

A crucial point for verification of universal or existential termination is to check feasibility of a symbolic run. Let us show how the feasibility of a symbolic run $\rho^S = C_0^S.C_1^S.\ldots.C_k^S$ is verified. The module checks the feasibility of a run ρ^S ending with an empty dataset by deriving backward weakest precondition from the deadlocked symbolic configuration to the initial symbolic configuration. At each symbolic configuration, the module derives the weakest precondition. The derived weakest precondition is then passed to Z3 solver to check the satisfiability. If the weakest precondition at a symbolic configuration C_i^S is not satisfiable, then it stops, and concludes that run ρ^S is not feasible with the condition ψ_k expected in its last configuration C_k^S. If we reach the initial configuration (all the derived weakest precondition FO formula along the run are satisfiable), then it remains to check that the input data verify the last precondition ψ_0 generated. If this is the case, then C_i^S can deadlock, and the module witnesses a deadlocked run resulting into non-termination of the complex workflow.

The property checker then builds on verification of symbolic runs realization to check existential termination. For universal termination, the verification module first verifies that rewritings rules do not contain cyclic dependencies. If it is the case the checker concludes immediately that the complex workflow does not terminate. If the specification is not recursive, then the checker can build the symbolic execution tree and verify individually each of its symbolic runs.

4.6 Conclusion

We studied termination and correctness of complex workflows with respect to the requirement on inputs and output of the overall process. Unsurprisingly, termination

of a complex workflow is undecidable, already due to the control part of the model. Now the question of whether all runs terminate can be answered when the initial data is specified in a fragment of FO for which satisfiability is decidable. Similar remarks apply to correctness. We consider the complexity of termination and correctness for different decidable FO fragments. The (co)-2EXPTIME bound for the fragments with the lowest complexity mainly comes from the exponential size of the formula depicting preconditions that must hold at initial configuration (the EXPTIME complexity is in the maximal length of runs). This can be seen as an untractable complexity, but one can however expect the depth of recursion to be quite low, or even enforce such a depth.

To summarize, complex workflows provide a way to rewrite an intricate task in an orchestration of smaller tasks and provide termination and correctness guarantees. However, focusing on the orchestration of complex workflows as coordination of smaller tasks is not enough. The complex workflow often comes with budget constraints provided by the client and also requires a guarantee on the data quality. The next two chapters will discuss the way to get reliable answers from the crowd with budget constraints. In particular, we use probabilistic models on top of workflows to obtain a trade-off between cost and accuracy.

PART III

Quality Assurance

CHAPTER 5

QUALITY ASSURANCE FOR ATOMIC TASKS

In the preceding chapters, we have proposed a *complex workflow* model that allows specifying intricate tasks with workflows. For this model, we can check correctness and termination on a reasonable subset of the language, mainly non-recursive specifications without dataset difference. Workflows provide an efficient way to execute intricate tasks to achieve a certain goal. They define the way tasks are decomposed, ordered, and executed. However, they do not provide mechanisms to guarantee the quality of data produced by the workflow. The verification process of Chapter 4 does not allow either to consider the cost of the workflow. Generally, tasks at crowdsourcing platforms come with a fixed budget provided by the client. In this chapter, we consider the cost and quality of produced data for a single atomic task. We extend the concept to workflows in Chapter 6.

The tasks at crowdsourcing platforms need human contribution. The simplest tasks include image annotation or classification, polling, etc. Employers publish tasks on an Internet platform, and these tasks are realized by workers in exchange for a small incentive [Dan+18]. Workers are very heterogeneous: they have different origins, domains of expertise, and expertise levels. For these reasons, workers can disagree and return very different answers, even for simple tasks. One can even consider malicious workers, that return wrong answers on purpose. To deal with this heterogeneity, tasks are usually replicated: each task is assigned to a *set of workers*. Redundancy is also essential to collect worker's opinion: in this setting, work units are the basic elements of a larger task that can be seen as a poll. One can safely consider that each worker executes his assigned task independently, and hence returns his own belief about the answer. As workers can disagree, the role of a platform is then to build a consensual final answer out of the values returned.

In Chapter 2, we have recalled a simple and natural way to derive a final answer using **Majority Voting** (MV). MV chooses as a conclusion the most represented answer. A limitation of MV is that all answers have equal weight, regardless of the expertise of workers. If a crowd is composed of only a few experts, and of a large majority of novices, MV favors answers from novices. However, in some domains, an expert worker may give a better answer than a novice and his answer should be given more weight. One can easily replace MV with a weighted vote. However, this raises the question of measuring a worker's expertise, especially when the worker's competencies are not known a priori. Crowdsourcing platforms such as Amazon Mechanical Turk (AMT) do not have prior knowledge about the expertise of their worker. A way to obtain an initial measure of a worker's expertise is to use **Golden Questions** [Le+10]. Several tasks with known *ground truth* are used explicitly or hidden to evaluate worker's expertise.

As already mentioned, a single answer for a particular task is often not sufficient to obtain a reliable answer, and one has to rely on redundancy, i.e. distribute the same task to several workers and aggregate results to build a final answer. Standard *static* approaches on crowdsourcing platforms fix a prior number of k workers per task. Each task is published on the platform and waits for bids by k workers. There is no guideline to set the value for k, but two standard situations where k is fixed are frequently met. The first case is when a client has n tasks to complete with a total budget of B_0 incentive units. Each task can be realized by $k = B_0/n$ workers. The second case is when an initial budget is not known, and the platform fixes an arbitrary redundancy level. In this case, the number of workers allocated to each task is usually between 3 and 10 [GM+16]. It is assumed that the distribution of work is uniform, i.e. that each task is assigned the same number of workers. An obvious drawback of static allocation of workers is that all tasks benefit from the same work power, regardless of their difficulty. Even a simple question where the variance of answers is high calls for a sampling of larger size. So, one could expect each task t to be realized by k_t workers, where k_t is a number that guarantees that the likelihood to change the final answer with the contribution of one additional worker is low. However, without prior knowledge on the task's difficulty and on variance in answers, this number k_t cannot be fixed a priori.

In this chapter, we propose an algorithm to address the questions of answers aggregation, task allocation, and costs optimization. We first propose an aggregation technique based on **Expectation Maximization (EM)** algorithm considering factors such as task difficulty and worker expertise. For simplicity, we consider boolean filtering tasks,

i.e. tasks with answers in $\{0, 1\}$, but the setting can be easily extended to tasks with any finite set of answers. These tasks are frequent, for instance, to decide whether a particular image belongs or not to a given category of pictures. We consider that each binary task has a *truth label*, i.e. there exists a ground truth for each task. Each worker is asked to answer 0 or 1 to such a task and returns a so-called *observed label*, which may differ from the ground truth. The *difficulty* of a task is a real value in $[0, 1]$. A task with difficulty 0 is a very easy task and a task with difficulty 1 a very complex one. The *expertise* of a worker is modeled in terms of *recall* and *specificity*. **Recall** (also called true positive rate) measures the proportion of correct observed labels given by a worker when the ground truth is 1. On contrary, **specificity** (also called true negative rate) measures the proportion of correct observed labels given by a worker when the ground truth is 0. In this chapter, we propose a generating function to model the probability of accuracy for each of the truth labels $(0/1)$ based on the *observed label, task difficulty, and worker expertise*. We rely on an Expectation Maximization (EM) based algorithm to estimate most probable ground truth for each task and jointly estimate the difficulty of each task as well as the expertise of the workers. The algorithm provides greater weight to expert workers. In addition, if a worker with high *recall* makes a mistake in the *observed label*, then it increases the difficulty of the task (correspondingly for specificity). Similarly, if expert workers fail to return a correct answer, then the task is considered difficult. The EM algorithm converges with a very low error rate and at the end returns the task *difficulty*, worker *expertise* and the *final estimated label* for each task based on *observed labels*.

We then propose *CrowdInc*, a dynamic worker allocation algorithm that handles at the same time aggregation of answers, and optimal allocation of a budget to reach a consensus among workers. The algorithm works in two phases. For the initial *Estimation* phase, as we do not have any prior information about the task difficulty and worker expertise, we allocate one-third of the total budget to inspect the behavior of each task. Based on the answers provided by human workers for each task, we first derive the difficulty of tasks, final aggregated answers, and the worker expertise using an EM algorithm. For each task, we estimate the likelihood that the aggregated answer is the ground truth. Tasks which aggregated answers have been synthesized with sufficient confidence are not allocated workers in the following rounds. Next phase is called *Convergence*. The remaining tasks are allocated additional workers for a new round based on the current estimate of the difficultly of tasks. At each round, we compute the task

difficulty, worker's expertise, and the final answer for each task, and then final answers with enough confidence are considered as the aggregated answer. This iterative process stops when all tasks are labeled with sufficient confidence, or when the budget is exhausted.

We provide the below outline of the chapter.

— We state the preliminaries and the factors affecting the aggregation techniques in Section 5.1.

— We use an Expectation Maximization based aggregation algorithm which aggregates the answers provided by the crowd workers and returns the most probable answer as the final answer. (Section 5.2).

— Next, we define a cost model that dynamically allocates workers based on task difficulty, confidence in the final answer, tasks to be processed, and the available budget. (Section 5.3).

— We evaluate the proposed model by comparing it with the state-of-the-art techniques on real datasets in Section 5.4 and finally conclude in Section 5.5.

5.1 Basic ingredients of aggregation

In this section, we detail the basic definitions of probability theory, factors influencing the efficiency of crowdsourcing, and the Expectation Maximization algorithm. We first introduce the concept of probability, conditional probability, and Bayes's law. We then discuss the two factors namely *difficulty* of the task and worker *expertise* that influence the aggregation mechanism. In the end, we present the generalized EM algorithm.

5.1.1 Probability theory

We work with discrete variables and discrete probabilities. A *random variable* is a variable whose value depends on random phenomenon called as event E. For a given variable x, we denote by $Dom(x)$ its domain (Boolean, integer, real, string, ...). For a particular value $v \in Dom(x)$ we denote by $x = v$, the event "x has value v". A probability measure $Pr(x = v)$ is a function that defines how likely a particular event is and takes a real value in $[0, 1]$. In the rest of the chapter, we mainly consider Boolean events, i.e. variables with domain $\{0, 1\}$. A probability of the form $Pr(x = v)$ only considers occurrence of a single event. When considering several events, we define the *joint*

probability. Joint probability is the occurrence of two events $E := (x = v)$ and $E' := (y = v')$ simultaneously and is denoted by $Pr(x = v, y = v')$. The notation extends to an arbitrary number of variables. If E and E' are two independent events, then their joint probability is $Pr(x = v, y = v') = Pr(x = v) \cdot Pr(y = v')$. Conditional Probability is defined as probability of an event when another event is known. More precisely, conditional probability is of the form $Pr(x = v|y = v')$ and is defined as the probability for an event $x = v$ given $y = v'$. The conditional probability is calculated as $Pr(x = v|y = v') = \frac{Pr(x=v,y=v')}{Pr(y=v')}$ assuming that $Pr(y = v') > 0$. Using all the notations, Bayes' law tells about the probability of an event based on some prior knowledge of conditions that might be relevant to the event. Considering A and B are events and $Pr(B) \neq 0$, then $Pr(A|B) = \frac{Pr(B|A) \times Pr(A)}{Pr(B)}$.

5.1.2 Factors influencing efficiency of crowdsourcing

During task labeling, several factors can influence the efficiency of crowdsourcing, and the accuracy of aggregated answers. The first one is **Task difficulty**. Tasks submitted to crowdsourcing platforms by a client address simple questions, but may nevertheless require some expertise. Even within a single application type, the difficulty for the realization of a particular task may vary from one experiment to another: tagging an image can be pretty simple if the worker only has to decide whether the picture contains an animal or an object, or conversely very difficult if the boolean question asks whether a particular insect picture shows a hymenopteran (an order of insects). Similarly, **Expertise of workers** plays a major role in the accuracy of aggregated answers. In general, an expert worker performs better on a specialized task than a randomly chosen worker without particular competence in the domain. For example, an entomologist can annotate an insect image more precisely than any random worker.

The technique used for **Amalgamation** also plays a major role. Given a set of answers returned for a task t, one can aggregate the results using *majority voting* (MV), or more interesting, as a weighted average answer where individual answers are pondered by worker's expertise. However, it is difficult to get a prior measure of worker's expertise and of the difficulty of tasks. Many crowdsourcing platforms use MV and ignore the difficulty of tasks and expertise of workers to aggregate answers or assign tasks to workers. We show in Section 5.4 that MV has low accuracy. We will also show in this chapter that expertise and difficulty, considered as hidden parameters can be

evaluated from the sets of answers returned. This allows us to hire new workers with a priori unknown expertise. One can also start with a priori measure of task difficulty and of worker's expertise. Worker's expertise can be known from former interactions. It is more difficult to have initial knowledge of task difficulties, but one can start with an a priori estimation. However, these measures need to be re-evaluated on the fly when new answers are provided by the crowd. Starting with a priori measures does not change the algorithms proposed hereafter, but may affect the final aggregated results.

In Section 5.2, we propose a technique to estimate the expertise of workers and difficulty of tasks on the fly. Intuitively, one wants to consider a task as difficult if even experts fail to provide a correct answer for this task and consider as it easy if even workers with low competence level answer correctly. Similarly, a worker is competent if he answers correctly difficult tasks. Notice however that to measure the difficulty of tasks and expertise of workers, one needs to have the final answer for each task. Conversely, to precisely estimate the final answer one needs to have the worker expertise and task difficulty. This is a chicken and egg situation, but we show in Section 5.2 how to get plausible value for both using EM.

The next issue to consider is the **cost** of crowdsourcing. Workers receive incentives for their work, but usually, clients have limited budgets. Some tasks may require a lot of answers to reach a consensus, while some may require only a few answers. Therefore, a challenge is to spend efficiently the budget to get the most accurate answers. In Section 5.3, we discuss some of the key factors in budget allocation. Many crowdsourcing platforms do not consider *difficulty*, and allocate the same number of workers to each task. The allocation of many workers to easy tasks is usually not justified and is a waste of budget that would be useful for difficult tasks. Now, tasks difficulty is not a priori known. This advocates for on the fly worker allocation once the difficulty of a task can be estimated. Last, one can stop collecting answers for a task when there is evidence that enough answers have been collected to reach a consensus on a final answer. An immediate solution is to measure the confidence of final aggregated answer and take as **Stopping Criterion** for a task the fact that this confidence exceeds a chosen threshold. However, this criterion does not work well in practice as clients usually want high thresholds for all their tasks. This may lead to consuming all available budget without reaching an optimal accuracy. In Section 5.3, we give a stopping criterion that balances confidence in the final answers and budget, and optimizes the overall accuracy of answers for all the tasks.

5.1.3 Expectation Maximization

Expectation Maximization [DLR77] is an iterative technique to obtain maximum likelihood estimation of the parameters of a statistical model when some parameters are unobserved and *latent*, i.e. they are not directly observed but rather inferred from observed variables. In some sense, the EM algorithm is a way to find the best fit between data samples and parameters. It has many applications in machine learning, data mining and Bayesian statistics.

Let $\mathcal{M}$ be a model which generates a set $\mathcal{X}$ of observed data, has a set of missing latent data $\mathcal{Y}$, and a vector of unknown parameters θ, along with a likelihood function $L(\theta \mid \mathcal{X}, \mathcal{Y}) = p(\mathcal{X}, \mathcal{Y} \mid \theta)$. Here, in a crowdsourcing context observed data $\mathcal{X}$ represents the answers provided by the crowd, $\mathcal{Y}$ depicts the *final answers* which need to be estimated and are hidden, and parameters in θ are the *difficulty* of tasks and the *expertise* of workers. The *maximum likelihood estimate* (MLE) of the unknown parameters is determined by maximizing the marginal likelihood of the observed data. We have $L(\theta \mid \mathcal{X}) = p(\mathcal{X} \mid \theta) = \int p(\mathcal{X}, \mathcal{Y} \mid \theta) d\mathcal{Y}$. The EM algorithm computes iteratively MLE, and proceeds in two steps. At the kth iteration of the algorithm, we let θ^k denote the estimate of parameters θ. At the first iteration of the algorithm, θ^0 is randomly chosen. Then the algorithm repeats two steps until convergence:

E-Step: In the E step, the missing data are estimated given observed data and the current estimate of parameters. The E-step computes the expected value of $L(\theta \mid \mathcal{X}, \mathcal{Y})$ given the observed data $\mathcal{X}$ and the current parameter θ^k. We define

$$Q(\theta \mid \theta^k) = \mathbb{E}_{\mathcal{Y} \mid \mathcal{X}, \theta^k}[L(\theta \mid \mathcal{X}, \mathcal{Y})] \tag{5.1}$$

In the crowdsourcing context, we use the E-Step to compute the probability of occurrence of $\mathcal{Y}$ that is the *final answer* for each task, given the observed data $\mathcal{X}$ and parameters θ^k obtained at k^{th} iteration.

M-Step: The M-step finds parameters θ that maximize the expectation computed in equation. 5.1.

$$\theta^{k+1} = \arg\max_{\theta} Q(\theta \mid \theta^k) \tag{5.2}$$

Here, with respect to estimated probability for $\mathcal{Y}$ for *final answers* from the last E-Step, we maximize the joint log-likelihood of the observed data $\mathcal{X}$ (answer provided by the

crowd), hidden data $\mathcal{Y}$ (final answers), to estimate the new value of θ^{k+1} i.e. the *difficulty* of tasks and the *expertise* of workers. The E and M steps are repeated until the value of θ^k converges. We present the algorithm more formally below (Algorithm 1) .

Algorithm 1: General EM Algorithm

Data: Observed Data $\mathcal{X}$, $Pr(\mathcal{X}, \mathcal{Y} \mid \theta)$
Result: Parameter values θ, Hidden data $\mathcal{Y}$
Initialize parameters in θ^0 to some random values.
while $||\theta^k - \theta^{k-1}|| > \epsilon$ **do**
    Compute the expected possible value of $\mathcal{Y}$, given θ^k and observed data $\mathcal{X}$
    Use $\mathcal{Y}$ to compute the values of θ that maximize $Q(\theta \mid \theta^k)$.
end
return parameter θ^k, Hidden data $\mathcal{Y}$

EM algorithm is widely studied and has diverse applications in estimation of mixed models [VD00; LB88], signal processing [Moo96], image processing [Car+02], machine learning [AA11; SDS20; TAK20], etc. We use EM as a black box algorithm. For a complete desription, we refer to the following references [Blu02; Moo96].

5.2 The Aggregation model

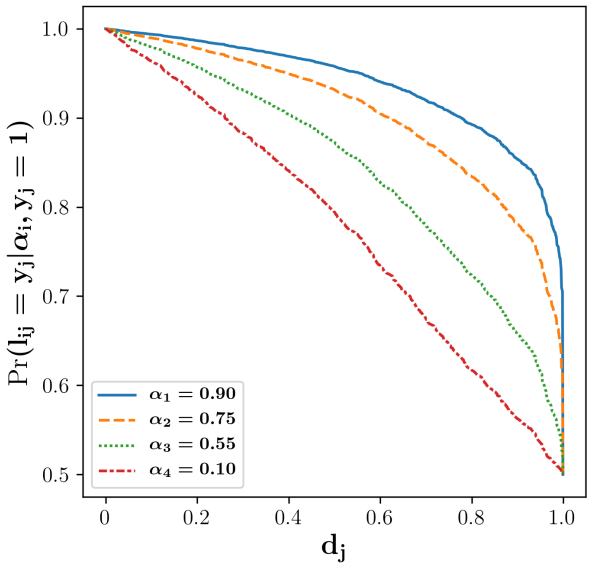
We address the problem of evaluation of binary properties of samples in a dataset by aggregation of answers returned by participants in a crowdsourcing system. For simplicity, we consider boolean tagging tasks, i.e. tasks with answers in $\{0, 1\}$, but the setting can be easily extended to tasks with any finite set of answers. This type of application is frequently met: one can consider for instance a database of n images, for which workers have to decide whether each image is clear or blur, whether a cat appears on the image, etc. The evaluated property is binary, i.e. worker's answers can be represented as a label in $\{0, 1\}$. From now, we will consider that tasks are elementary work units whose objective is to associate a binary label to a particular input object. For each task, an actual ground truth exists, but it is not known by the crowdsourcing platform. We assume a set of k independent workers, which role is to realize a task, i.e. return an *observed label* in $\{0, 1\}$ according to their perception of a particular sample. We consider a set of tasks $T = \{t_1, \dots t_n\}$ for which a label must be evaluated. For a task $t_j \in T$ the observed label given by worker $1 \leq i \leq k$ is denoted by

l_{ij}. We let y_j denote the *final label* of a task t_j obtained by aggregating the answers of all workers. $L_j = \bigcup_{i \in 1..k} l_{ij}$ denotes the set of all labels returned by workers for task t_j, L denotes the set of all observed labels, $L = \bigcup_{j \in 1..n} L_j$. The goal is to estimate the ground truth by synthesizing a set of *final labels* $Y = \{y_j, 1 \leq j \leq n\}$ from the set of *observed label* L for all tasks.

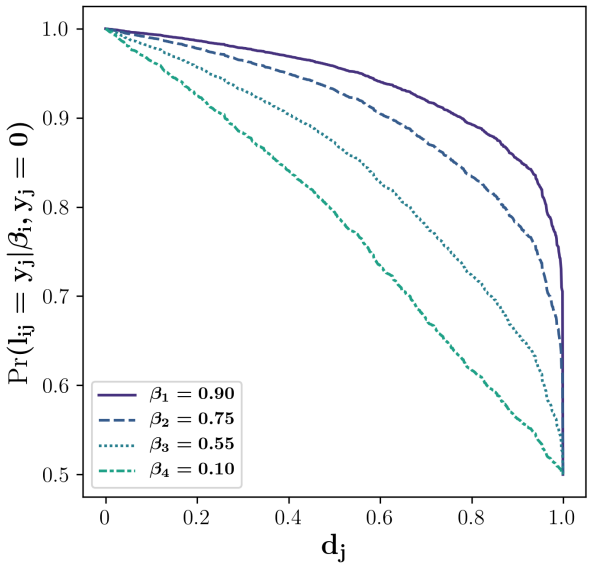
Despite the apparent simplicity of the problem, crowdsourcing binary tagging tasks hide several difficulties, originating from unknown parameters. These parameters are the difficulty of each task, and the expertise of each worker. The difficulty of task t_j is modeled by a parameter $d_j \in [0, 1]$. Here value 0 means that the task is very easy, and can be performed successfully by any worker. On the other hand, $d_j = 1$ means that task t_j is very difficult. A standard way to measure expertise is to define workers accuracy as a pair $\xi_i = \{\alpha_i, \beta_i\}$, where α_i is called the *recall* of worker i and β_i the *specificity* of worker i. The **recall** is the probability that worker i annotates an image j with label 1 when the ground truth is 1, i.e. $\alpha_i = Pr(l_{ij} = 1 | y_j = 1)$. The **specificity** of worker i is the probability that worker i annotates an image j with 0 when the ground truth is 0, i.e. $\beta_i = Pr(l_{ij} = 0 | y_j = 0)$.

In literature, [Zhe+17] the expertise of workers is often quantified in terms of *accuracy*, i.e. $Pr(l_{ij} = y_j)$. However, if the data samples are unbalanced, i.e. the number of samples with actual ground truth 1 (respectively 0) is much larger than the number of samples with ground truth 0 (respectively 1), defining competencies in terms of *accuracy* leads to bias. Indeed, a worker who is good at classifying images with ground truth 1 can obtain bad scores when classifying images with ground truth 0, and yet get a good accuracy (this can be the case of a worker that always answers 1 when tagging a task). *Recall* and *Specificity* overcomes the problem of bias and separates the worker expertise, considering their ability to answer correctly when the ground truth is 0 and when it is 1, and hence give a more precise representation of workers competences.

The exact behavior of workers is not exactly known. However, it is clear that the probability to answer correctly an easy task is higher than the probability to answer correctly a difficult one. We can hence build a probabilistic model (a generative model) to estimate worker's answers. We assume that workers have constant behaviors and are faithful, i.e. do not return wrong answers intentionally. We also assume that workers do not collaborate (their answers are independent variables). Under these assumptions, knowing the recall α_i and specificity β_i of a worker i, we build a model that generates the probability that the worker i returns an answer l_{ij} for a task j with difficulty d_j.



(a) Generative function for the probability to get $l_{ij} = 1$, given $y_j = 1$, for growing values of task difficulty. The curves represent different values of recall for the considered workers.



(b) Generative function for the probability to get $l_{ij} = 0$, given $y_j = 0$, for growing values of task difficulty. The curves represent different values of specificity for the considered workers.

Figure 5.1 – Generating function.

$$Pr(l_{ij} = y_j | d_j, \alpha_i, y_j = 1) = \frac{1 + (1 - d_j)^{(1-\alpha_i)}}{2} \tag{5.3}$$

$$Pr(l_{ij} = y_j | d_j, \beta_i, y_j = 0) = \frac{1 + (1 - d_j)^{(1-\beta_i)}}{2} \tag{5.4}$$

This model is defined by equations 5.3 and 5.4, that characterize respectively the probability to get a correct answer for a given recall α_i when ground truth is 1 and for a given specificity β_i when ground truth is 0. Figure 5.1a shows the probability of associating label 1 to a task for which the ground truth is 1 when the difficulty of the tagging task varies and for different values of recall. The range of task difficulty is $[0, 1]$. The vertical axis is the probability of getting $l_{ij} = 1$. One can notice that this probability takes values between 0.5 and 1. Indeed, if a task is too difficult, then returning a value is close to making a random guess of a binary value. Unsurprisingly, as the difficulty of the task increases, the probability of correctly labeling the task decreases. For a fixed difficulty of the task, workers with higher recalls have a higher probability to correctly

label a task. Also, note that when the difficulty of a task approaches 1, the probability of answering with label $l_{ij} = 1$ decreases for every value of α_j. However, for workers with high recall, the probability of a correct annotation is always greater than with a smaller recall. Hence, the probability of a correct answer depends both on the difficulty of the task and on the expertise of the worker realizing the task. Similarly, Figure 5.1b represents the probability of tagging a task to a label 0 for which the ground truth is 0 with varying difficulty of the tagging task and different values of specificity.

5.2.1 Aggregating Answers

For a given task j, with unknown difficulty d_j, the answers returned by k workers (observed data) is a set $L_j = \{l_{1j}, \ldots, l_{kj}\}$, where l_{ij} is the answer of worker i to task j. In addition, the expertise of k workers is defined by the pair of vectors of parameters $\alpha = \{\alpha_1, \ldots, \alpha_k\}$ and $\beta = \{\beta_1, \ldots, \beta_k\}$ and are also unknown. The goal of a crowdsourcing platform is to infer the final label y_j, and to derive the most probable values for d_j, α_i, β_i, given the observed answers of workers. We use a standard EM approach to infer the most probable actual answer $Y = \{y_1, \ldots y_n\}$ along with the hidden parameters $\theta = \{d_j, \alpha_i, \beta_i \mid j \in 1 \cdots n \wedge i, j \in 1 \cdots k\}$. Let us consider the E and M phases of the algorithm.

E Step: We assume that all answers in $L = \bigcup_{1 \leq j \leq m} L_j$ are independently given by the workers as there is no collaboration between them. So, in every $L_j = \{l_{1j}, \ldots, l_{kj}\}$, l_{ij}'s are independently sampled variables. We compute the posterior probability of $y_j \in \{0, 1\}$ for a given task j given the difficulty of task d_j, worker expertise $\alpha_i, \beta_i, i \leq k$ and the worker answers $L_j = \{l_{ij} \mid i \in 1..k\}$. Using Bayes' theorem, considering a particular value $\lambda \in \{0, 1\}$ we have:

$$Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j) \cdot Pr(y_j = \lambda | \alpha, \beta, d_j)}{Pr(L_j | \alpha, \beta, d_j)} \tag{5.5}$$

The value of the final label y_j directly depends upon the answers L_j provided by the workers. Therefore, y_j and α, β, d_j are independent variables. We assume that prior probability of y_j taking values as 0 or 1 is equiprobable, i.e. the prior probability to choose between the final answers are $Pr(y_j = 0) = Pr(y_j = 1) = \frac{1}{2}$. We hence get:

$$Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j) \cdot Pr(y_j = \lambda)}{Pr(L_j | \alpha, \beta, d_j)} = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j) \cdot \frac{1}{2}}{Pr(L_j | \alpha, \beta, d_j)} \tag{5.6}$$

Similarly, the probability to obtain a particular set of labels is given by:

$$Pr(L_j \mid \alpha, \beta, d_j) = \frac{1}{2} \cdot Pr(L_j \mid y_j = 0, \alpha, \beta, d_j) + \frac{1}{2} \cdot Pr(L_j \mid y_j = 1, \alpha, \beta, d_j) \quad (5.7)$$

Overall we obtain:

$$Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j)}{Pr(L_j | y_j = 0, \alpha, \beta, d_j) + Pr(L_j | y_j = 1 \alpha, \beta, d_j)} \quad (5.8)$$

Let us consider one of these terms, and let us assume that every l_{ij} in L_j takes a value $\lambda_{p_{ij}}$. We have

$$Pr(L_j \mid y_j = \lambda, \alpha, \beta, d_j) = \prod_{i=1}^{k} Pr(l_{ij} = \lambda_p \mid \alpha_i, \beta_i, d_j, y_j = \lambda) \quad (5.9)$$

If $\lambda_{p_{ij}} = 0$ then $Pr(l_{ij} = \lambda_{p_{ij}} \mid \alpha_i, \beta_i, d_j, y_j = 0)$ is the probability to classify correctly a 0 as 0, as defined in equation 5.4 denoted by $\delta_{ij} = \frac{1+(1-d_j)^{(1-\beta_i)}}{2}$. Similarly, if $\lambda_{p_{ij}} = 1$ then $Pr(l_{ij} = \lambda_{p_{ij}} \mid \alpha_i, \beta_i, d_j, y_j = 1)$ is the probability to classify correctly a 1 as 1, expressed in equation 5.3 and denoted by $\gamma_{ij} = \frac{1+(1-d_j)^{(1-\alpha_i)}}{2}$. Then the probability to classify $y_j = 1$ as $\lambda_{p_{ij}} = 0$ is $(1 - \gamma_{ij})$ and the probability to classify $y_j = 1$ as $\lambda_{p_{ij}} = 0$ is $(1 - \delta_{ij})$. We hence have $Pr(l_{ij} = \lambda_{p_{ij}} \mid \alpha_i, \beta_i, d_j, y_j = 0) = (1 - \lambda_{p_{ij}}) \cdot \delta_{ij} + \lambda_{p_{ij}} \cdot (1 - \gamma_{ij})$. Similarly, we can write $Pr(l_{ij} = \lambda_{p_{ij}} \mid \alpha_i, \beta_i, d_j, y_j = 1) = \lambda_{p_{ij}} \cdot \gamma_{ij} + (1 - \lambda_{p_{ij}}) \cdot (1 - \delta_{ij})$. So equation 5.8 rewrites as :

$$\begin{aligned} Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] &= \frac{\prod_{i=1}^{k} Pr(l_{ij} = \lambda_{p_{ij}} \mid y_j = \lambda_{p_{ij}}), \alpha_i, \beta_i, d_j}{Pr(L_j \mid y_j = 0, \alpha, \beta, d_j) + Pr(L_j \mid y_j = 1, \alpha, \beta, d_j)} \\ &= \frac{\prod_{i=1}^{k} (1 - \lambda_{p_{ij}}).[(1 - \lambda_{p_{ij}})\delta_{ij} + \lambda_{p_{ij}}(1 - \gamma_{ij})] + \lambda_{p_{ij}}[\lambda_{p_{ij}}.\gamma_{ij} + (1 - \lambda_{p_{ij}})(1 - \delta_{ij})]}{Pr(L_j \mid y_j = 0, \alpha, \beta, d_j) + Pr(L_j \mid y_j = 1, \alpha, \beta, d_j)} \\ &= \frac{\prod_{i=1}^{k} (1 - \lambda_{p_{ij}}).[(1 - \lambda_{p_{ij}})\delta_{ij} + \lambda_{p_{ij}}(1 - \gamma_{ij})] + \lambda_{p_{ij}}.[\lambda_{p_{ij}}.\gamma_{ij} + (1 - \lambda_{p_{ij}})(1 - \delta_{ij})]}{\prod_{i=1}^{k} (1 - \lambda_{p_{ij}})\delta_{ij} + \lambda_{p_{ij}}(1 - \gamma_{ij}) + \prod_{i=1}^{k} \lambda_{p_{ij}}.\gamma_{ij} + (1 - \lambda_{p_{ij}})(1 - \delta_{ij})} \end{aligned} \quad (5.10)$$

In the E step, as every α_i, β_i, d_j is fixed, one can compute $\mathbb{E}[y_j | L_j, \alpha_i, \beta_i, d_j]$ and also choose as final value for y_j the value $\lambda_j \in \{0, 1\}$ such that $Pr[y_j = \lambda_j | L_j, \alpha_i, \beta_i, d_j] \geq Pr[y_j = (1 - \lambda_j) | L_j, \alpha_i, \beta_i, d_j]$. We can also estimate the likelihood for the values of variables $P(L \cup Y \mid \theta)$ for parameters $\theta = \{\alpha, \beta, d\}$, as $Pr(y_j = \lambda_j, L \mid \theta) = Pr(y_j = \lambda_j, L).Pr(L_j \mid y_j = \lambda_j, \theta) = Pr(y_j = \lambda_j).Pr(L_j \mid y_j = \lambda_j, \theta)$

M Step: With respect to the estimated posterior probabilities of Y computed during the E phase of the algorithm, we compute the parameters θ that maximize $Q(\theta, \theta^t)$.

Let θ^t be the value of parameters computed at step t of the algorithm. We use the observed values of L, and the previous expectation for Y. We maximize $Q'(\theta, \theta^t) = \mathbb{E}[logPr(L, Y \mid \theta) \mid L, \theta^t]$ (we refer to [Fla12]-Chap. 9 and [DLR77] for explanations showing why this is equivalent to maximizing $Q(\theta, \theta^t)$). We can hence compute the next value as: $\theta^{t+1} = \arg\max_{\theta} Q'(\theta, \theta^t)$. Here in our context the values of θ are α_i, β_i, d_j. We maximize $Q'(\theta, \theta^t)$ using a bounded optimization technique, namely truncated Newton algorithm [Nas84; NW06] provided by the standard SciPy [1] implementation. We iterate E and M steps, computing at each iteration t the posterior probability and the parameters θ^t that maximize $Q'(\theta, \theta^t)$. The algorithm converges, and stops when the improvement (difference between two successive joint log-likelihood values) is below a threshold, fixed in our case to $1e^{-7}$.

5.3 Cost Model

A drawback of many crowdsourcing approaches is that task distribution is static [GM+16], i.e. tasks are distributed to a fixed number of workers, without considering their difficulty, nor checking if a consensus can be reached with fewer workers. Consider again the simple boolean tagging setting, but where each task realization is paid and with a fixed maximal budget B_0 provided by the client. For simplicity, we assume that all workers receive 1 unit of credit for each realized task. Hence, to solve n boolean tagging tasks, one can hire at most B_0/n workers per task. In this section, we show a worker allocation algorithm that builds on collected answers and estimated difficulty to distribute tasks to workers at run time and show its efficiency with respect to other approaches.

The algorithm works in rounds. At each round, only a subset $T_{avl} \subseteq T$ of the initial tasks remain to be evaluated. We collect labels produced by workers for these tasks. We aggregate answers using the EM approach described in Section 5.2. We denote by y_j^q as the final aggregated answer for task j at round q, d_j^q is the current *difficulty* of task and α_i^q, β_i^q denotes the estimated *expertise* of a worker i at round q. We let $D^q = \{d_1^q \dots d_j^q\}$ denote the set of all difficulties estimated as round q. We fix a maximal step size $\tau \geq 1$, that is the maximal number of workers that can be hired during a round for a particular task. For every task $t_j \in T_{avl}$ with difficulty d_j^q at round q, we allocate $\mathbf{a}_j^q = \lceil (d_j^q / \max D^q) \times \tau \rceil$ workers for the next round. Once all answers for a task have been received, the EM aggregation can compute final label $y_j^q \in \{0, 1\}$, difficulty of task

1. docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

d_j^q, expertise of all workers $\alpha_1^q, \ldots, \alpha_k^q, \beta_1^q, \ldots, \beta_k^q$. At the end of each round, for each task t_j one has to decide whether the **confidence** in answer y_j^q obtained at round q is sufficient (in which case, we do not allocate workers to this task in the next rounds, i.e. we remove it from T_{avl}).

5.3.1 Confidence and Threshold

An important point is to estimate confidence. Another crucial point is to define the confidence threshold at each round q of the algorithm. Let k_j^q be the number of answers obtained for task j at round q. The *confidence* $\hat{c}_j^q$ in a final label y_j^q is defined as follows:

$$\hat{c}_j^q(y_j^q = 1) = \frac{1}{k_j^q} \cdot \sum_{i=1}^{k_j^q} \left\{ l_{ij} \times \left(\frac{1+(1-d_j^q)^{(1-\alpha_i^q)}}{2}\right) + (1 - l_{ij}) \times \left(1 - \frac{1+(1-d_j^q)^{(1-\alpha_i^q)}}{2}\right) \right\} \quad (5.11)$$

$$\hat{c}_j^q(y_j^q = 0) = \frac{1}{k_j^q} \cdot \sum_{i=1}^{k_j^q} \left\{ (1 - l_{ij}) \times \left(\frac{1+(1-d_j^q)^{(1-\beta_i^q)}}{2}\right) + (l_{ij}) \times \left(1 - \frac{1+(1-d_j^q)^{(1-\beta_i^q)}}{2}\right) \right\} \quad (5.12)$$

Intuitively, each worker adds its probability of doing an error, which depends on the final label y_j^q estimated at round q and on his competences, i.e. on the probability to choose $l_{ij} = y_j^q$. Let us now show when to stop the rounds of our evaluation algorithm. We start with n tasks, and let T_{avl} denote the set of remaining tasks at round q. We define $r^q \in [0, 1]$ as the ratio of tasks that are still considered at round q compared to the initial number of task, i.e. $r^q = \frac{|T_{avl}|}{n}$. We start with an initial budget B_0, and denote by B_c^q the total budget consumed at round q. We denote by $\mathcal{B}^q$ the the fraction of budget consumed at that current instance, $\mathcal{B}^q = \frac{B_c^q}{B_0}$. We define the stopping threshold $Th^q \in [0.5, 1.0]$ as $Th^q = \frac{1+(1-\mathcal{B}^q)^{r^q}}{2}$.

The intuition behind this function is simple: when the number of remaining tasks decreases, one can afford a highest confidence threshold. Similarly, as the budget decreases, one shall derive a final answer for tasks faster, possibly with poor confidence, as the remaining budget does not allow hiring many workers. Figure 5.2 shows how the threshold evolves for different values of r^q when the fraction of the budget consumed $\mathcal{B}^q$ evolves. Each line depicts the evolution of the threshold for different values of r^q. Observe that when r^q approaches 1, the threshold value falls rapidly, as a large number of tasks remain without a definite final answer and have to be evaluated within the remaining budget. On the other hand, when fewer tasks remain, (e.g. when $r^q = 0.10$), the threshold Th^q decreases slowly.

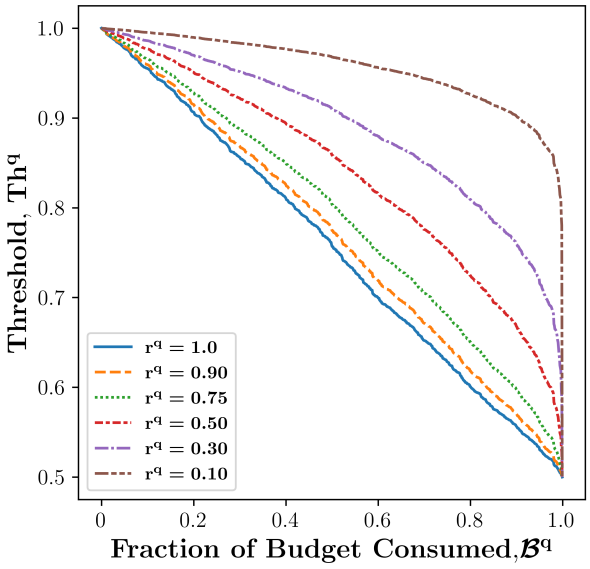


Figure 5.2 – The threshold values based on current estimate on consumed budget and fraction of task remaining at the beginning of a round.

5.3.2 CrowdInc: An algorithm to optimize costs

We now propose a crowdsourcing algorithm called **CrowdInc** with a dynamic worker allocation strategy to optimize cost and accuracy. This strategy allocates workers depending on current confidence on final answers, and available resources. CrowdInc is decomposed in two phases, *Estimation* and *Convergence*.

Estimation: As *difficulty* of tasks is not known a priori, the first need is to estimate it. To get an initial measure of difficulties, each task needs to be answered by a small set of workers. Now, as each worker receives an incentive for a task, this preliminary evaluation has a cost, and finding an optimal number of workers for *difficulty* estimation is a fundamental issue. The initial budget B_0 gives some flexibility in the choice of an appropriate number of workers for preliminary evaluation of difficulty. Choosing a random number of workers per task does not seem a wise choice. We choose to devote a fraction of the initial budget to this estimation phase. We devote one-third of the total budget ($B_0/3$) to the estimation phase and allocate each task $\zeta = (B_0/3)/n$ workers. It leaves a sufficient budget ($2 \cdot B_0/3$) for the convergence phase. Experiments in the next Section 5.4 show that this seems a sensible choice. After the collection of answers for each task, we apply the EM-based aggregation technique of Section 5.2 to estimate the

Algorithm 2: CrowdInc

Data: A set of tasks $T = \{t_1, \ldots, t_n\}$, a budget = B_0
Result: Final Answer: $Y = y_1, \ldots, y_n$, Difficulty: d_j,Expertise: α_i, β_i

Initialization : Set every d_j, α_i, β_i to a random value in $[0, 1]$.
$T_{avl} = T$; $q = 0$; $B = B - (B_0/3)$; $B_c = B_0/3$; $\zeta = (B_0/3)/n$
`//Initial Estimation:`
Allocate ζ workers to each task in T_{avl} and get their answers
Estimate $d_j^q, \alpha_i^q, \beta_i^q, \hat{c}_j^q, 1 \leq j \leq n, 1 \leq i \leq \zeta$ using EM aggregation
Compute the stopping threshold Th^q.
for $j = 1, \ldots, n$ **do**
    if $\hat{c}_j^q > Th^q$ **then** $T_{avl} = T \setminus \{j\}$;
end
`//Convergence:`
while $(B > 0)$ && $(T_{avl} \neq \emptyset)$ **do**
    $q = q + 1$; $l = |T_{avl}|$
    Allocate $\mathbf{a}_1^q, \ldots, \mathbf{a}_l^q$ workers to tasks $t_1, \ldots t_l$ based on difficulty.
    Get the corresponding answers by all the newly allocated workers.
    Estimate $d_j^q, \alpha_i^q, \beta_i^q, \hat{c}_j^q$ using aggregation model.
    $B = B - \sum_{i \in 1..|T_{avl}|} \mathbf{a}_i^q$
    Compute the stopping threshold Th^q
    for $j = 1, \ldots, n$ **do**
        if $\hat{c}_j^q > Th^q$ **then** $T_{avl} = T_{avl} \setminus \{j\}$;
    end
end

difficulty of each task as well as the *expertise* of each worker. Considering this as an initial round $q = 0$, we let d_j^0 denote the initially estimated difficulty of each task j, α_i^0, β_i^0 denote the expertise of each worker and y_j^0 denote the aggregated answer at round $q = 0$. Note that if the difficulty of some tasks is available a priori and is provided by the client, we may skip the estimation step. However, in general, clients do not possess such information and this initial step is crucial in the estimation of parameters. This is especially true when clients needs are to execute huge batches of tasks: attaching a prior difficulty with each task would be as costly as executing the tasks. After this initial estimation, one can already compute Th^0 as shown in Section 5.3.1 and decide to stop the evaluation of tasks with a sufficient confidence level.

Convergence: The *difficulty* of task d_j^q and the set of remaining tasks T_{avl} are used to start the convergence phase. Now as the difficulty of each task is estimated, we can use the estimated difficulty of d_j^q to allocate the workers dynamically. The number of workers allocated at round $q > 0$ follows a difficulty-aware *worker allocation* policy. We fix a parameter to allocate workers for the realization of tagging tasks at each round that allocates $\mathbf{a}_j^q = \lceil (d_j^q / \max D^q) \times \tau \rceil$ workers to each remaining task $t_j \in T_{avl}$. This allocation policy guarantees that each remaining task is allocated at least one worker, at most τ workers, and that the more difficult tasks (i.e. have the more disagreement) are allocated more workers than easier tasks. Algorithm 2 gives a full description of CrowdInc.

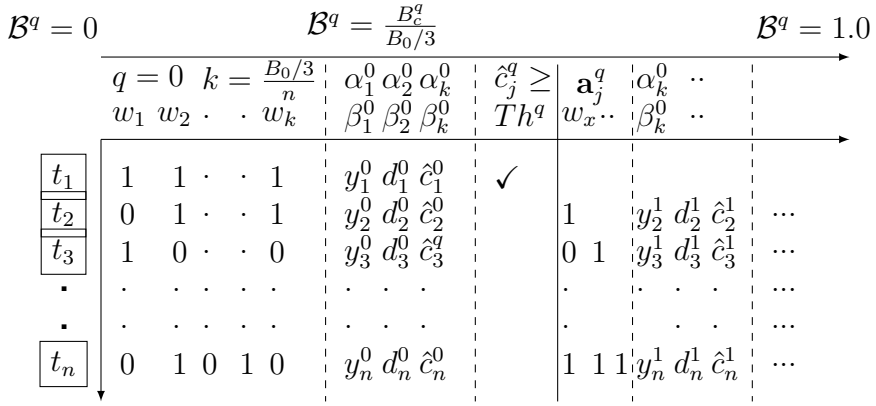


Figure 5.3 – A possible state for algorithm 2.

Example: We show an example depicting the information memorized at each step of the algorithm in Figure 5.3. Consider a set of n tasks that to be annotated with a boolean tag in $\{0, 1\}$. CrowdInc starts with the *Estimation* phase and allocates k workers for an initial evaluation round ($q = 0$). After collection of answers, at each round $q > 0$, we first apply EM based aggregation to estimate the difficulty d_j^q of each task $t_j \in T_{avl}$, the confidence $\hat{c}_j^q$ in aggregated answer y_j^q synthesized at round q, and the expertise α_i^q, β_i^q of the workers. Then, we use the stopping threshold to decide whether we need more answers for each task. If $\hat{c}_j^q$ is greater than Th^q, the task t_j is removed from T_{avl}. This stopping criterion hence takes a decision based on the confidence in the final answers for a task and on the remaining budget. Consider, in the example of Figure 5.3 that the aggregated answer for task t_1 has high confidence, and that $\hat{c}_1^q \geq Th^q$. Then, t_1 does not need further evaluation, and is removed from T_{avl}. Once solved tasks have been removed, we allocate $\mathbf{a}_j^q$ workers to each remaining task t_j in T_{avl} following our

Dataset	Number of Tasks	Number of tasks with ground truth	Total Number of answers provided by crowd	Average number of answers for each task	Number of unique crowd workers
Product Identification	8315	8315	24945	3	176
Duck Identification	108	108	4212	39	39
Sentiment Popularity	500	500	10000	20	143

Table 5.1 – Datasets description.

difficulty aware policy. Note that, each task gets a different number of workers based on its difficulty. The algorithm stops when either all budget is exhausted or there is no additional task left. It returns the aggregated answers for all tasks along with difficulty of task and the expertise of the workers. Notice that termination is guaranteed, as all n tasks receive at least $(B_0/3)/n$ answers and as the budget necessarily decreases.

5.4 Experiments

We evaluated the algorithm on three datasets, namely the product identification [Wan+12], duck identification [Wel+10] and Sentiment Analysis [PL04] benchmarks. We briefly detail each dataset and the corresponding tagging tasks. All tags appearing in the benchmarks were collected via Amazon Mechanical Turk.

In the **Product Identification** use case, workers were asked to decide whether a *product-name* and a *description* refer to the same product. The answer returned is *True* or *False*. There are 8315 samples and each of them was evaluated by 3 workers. The total number of unique workers is 176 and the total number of answers available is 24945. In the **Duck Identification** use case, workers had to decide if sample images contain a duck. The total number of tasks is 108 and each task was allocated to 39 workers. The total number of unique workers is 39 and the total number of answers is 4212. In the **Sentiment Popularity** use case, workers had to annotate movie reviews as Positive or Negative opinions. The total number of tasks was 500. Each task was given to 20 unique workers and a total number of 143 workers were involved, resulting in a total number of 10000 answers. All this information are summarized in Table 5.1.

Evaluation of aggregation: We first compared our aggregation technique to several methods: Majority Voting (MV), D&S [DS79], GLAD [Whi+09], PMCRH [Li+14], LFC [Ray+10], and ZenCrowd [DDCM12]. We ran the experiment 30 times with different initial values for tasks difficulty and workers expertise. The standard deviation

Methods	**Recall**	**Specificity**	**Balanced Accuracy**
MV	0.56	0.91	0.73
D&S [DS79]	0.81	0.93	0.87
GLAD [Whi+09]	0.47	0.98	0.73
PMCRH [Li+14]	0.58	0.95	0.76
LFC [Ray+10]	0.87	0.91	0.89
ZenCrowd [DDCM12]	0.39	0.98	0.68
EM + recall, specificity & difficulty	**0.89**	**0.91**	**0.90**

Table 5.2 – Comparison of EM + aggregation (with Recall, specificity & task difficulty) w.r.t MV, D&S, GLAD, PMCRH, LFC, ZenCrowd on *Duck Identification* dataset

Methods	**Recall**	**Specificity**	**Balanced Accuracy**
MV	0.61	0.93	0.77
D&S [DS79]	0.65	0.97	0.81
GLAD [Whi+09]	0.48	0.98	0.73
PMCRH [Li+14]	0.61	0.93	0.77
LFC [Ray+10]	0.64	0.97	0.81
ZenCrowd [DDCM12]	0.51	0.98	0.75
EM + recall, specificity & difficulty	**0.77**	**0.90**	**0.83**

Table 5.3 – Comparison of EM + aggregation (with Recall, specificity & task difficulty) w.r.t MV, D&S, GLAD, PMCRH, LFC, ZenCrowd on *Product Identification* dataset

Methods	Recall	Specificity	Balanced Accuracy
MV	0.93	0.94	0.94
D&S [DS79]	0.94	0.94	0.94
GLAD [Whi+09]	0.94	0.94	0.94
PMCRH [Li+14]	0.93	0.95	0.94
LFC [Ray+10]	0.94	0.94	0.94
ZenCrowd [DDCM12]	0.94	0.94	0.94
EM + recall, specificity & difficulty	**0.94**	**0.95**	**0.94**

Table 5.4 – Comparison of EM + aggregation (with Recall, specificity & task difficulty) w.r.t MV, D&S, GLAD, PMCRH, LFC, ZenCrowd on *Sentiment Popularity* dataset

Dataset/Methods	CrowdInc	Static(EM)	Static(MV)
Duck Identification	843.26	106.81	0.073
Sentiment Popularity	1323.35	137.79	0.102

Table 5.5 – Running time(in seconds) of CrowdInc, MV and Static EM.

over all the iteration was less than 0.05%. Hence our aggregation is insensitive to initial prior values. We now compare *Recall*, *Specificity* and *Balanced Accuracy* of all methods. Balanced accuracy (interchangeably we use accuracy) is the average of recall and specificity. Our method outperforms other techniques as shown in table 5.2 for Duck Identification and in table 5.3 for Product Identification dataset and is comparable for Sentiment Popularity dataset shown in table 5.4.

Evaluation of CrowdInc: The goal of the next experiment was to verify that the cost model proposed in CrowdInc achieves at least the same accuracy but with a smaller budget. We have used Duck identification and Sentiment popularity for this test. We did not consider the Product Identification benchmark: indeed, as shown in table 5.1, the Product Identification associates only 3 answers to each task. This does not allow for a significant experiment with CrowdInc. We compared the performance of CrowdInc to other approaches in terms of cost and accuracy. The results are given in Figure 5.4. Static(MV) denotes the traditional crowdsourcing platforms with majority voting as the aggregation technique and Static(EM) shows more advanced aggregation technique with EM based aggregation technique. Both algorithms allocate all the workers (and hence use all their budget) at the beginning of the crowdsourcing process.

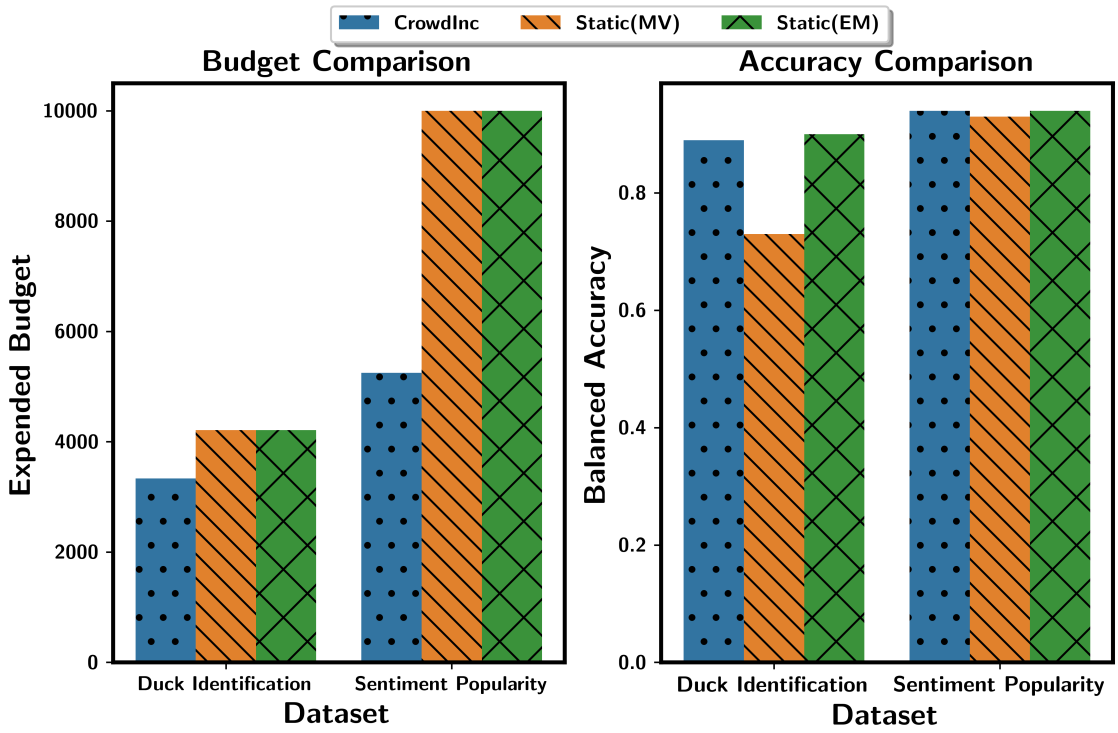


Figure 5.4 – Comparison of cost vs. Accuracy.

The following observation can be made from Figure 5.4. First, CrowdInc achieves better accuracy than a static(MV) approach. This is not a real surprise, as MV already showed bad accuracy (refer table 5.2, 5.3, 5.4). Then, CrowdInc achieves almost the same accuracy as a Static(EM) based approach in Duck identification, and the same accuracy in Sentiment Popularity. Last, CrowdInc uses a smaller budget than static approaches in all cases.

Table 5.5 shows the time (in seconds) needed by each algorithm to aggregate answers. Static(MV) is the fastest solution: it is not surprising, as the complexity is linear in the number of answers. We recall however that MV has the worst accuracy of all tested aggregation techniques. We have tested aggregation with EM when the number of workers is fixed a priori and is the same for all tasks (Static(EM)). CrowdInc uses EM, but on a dynamic set of workers and tasks, stopping the easiest tasks first. This results in a longer calculus, as EM is used several times on sets of answers of growing sizes. The accuracy of *static(EM)* and CrowdInc are almost the same. Aggregation with CrowdInc takes approximately 11% longer than *static(EM)* but for a smaller budget, as shown in Figure 5.4. To summarize the CrowdInc aggregation needs more time but a

smaller budget to aggregate answers with comparable accuracy. In general, clients using crowdsourcing services can wait several days to see their task completed. Hence, when time is not a major concern, CrowdInc can reduce the cost of crowdsourcing.

5.5 Conclusion

In this chapter, we introduced an aggregation technique for crowdsourcing platforms. Aggregation is based on expectation maximization algorithm that jointly estimates the answers, the difficulty of tasks, and the expertise of workers. Introducing new variables such as difficulty, and expertise of workers improve the accuracy of aggregation in terms of recall and specificity. We also proposed CrowdInc an incremental labeling technique that optimizes the cost of answers collection. The algorithm implements a worker allocation policy that takes decisions from a dynamic threshold computed at each round, which helps to achieve a trade-off between cost and accuracy. We showed in experiments that CrowdInc outperforms the existing state-of-the-art techniques. We also showed that the incremental crowdsourcing approach achieves the same accuracy as EM with the static allocation of workers, better accuracy than majority voting, and in both cases at lower costs.

The ideas proposed in this chapter can lead to several improvements that can be considered in future work. We addressed binary tasks for simplicity, but the approach can be easily extended to tasks with a finite number of m of answers. The difficulty of each task t_j remains a parameter $d_j \in [0, 1]$. Expertise becomes the ability to classify a task as m when its ground truth is m. An EM algorithm just has to consider probabilities of the form $Pr(L_{ij} = m|y_j = m)$ to derive hidden parameters and final labels for each task. An easy improvement is to consider incentives that depend on worker's characteristics. This can be done with a slight adaptation of costs in the CrowdInc algorithm. Another possible improvement is to try to hire experts when the synthesized difficulty of a task is high, to avoid hiring numerous workers or increase the number of rounds. Another interesting topic to consider is the impact of answers introduced by a malevolent worker on the final aggregated results. Last, we think that the complexity of CrowdInc can be improved. The complexity of each E-step of the aggregation is linear in the number of answers. The M-step maximizes the log-likelihood with an iterative process. However, the E and M steps have to be repeated many times. The cost of

this iteration can be seen in table 5.5, where one clearly sees the difference between an approach with linear time complexity such as Majority Voting (third column), a single round of EM (second column), and CrowdInc. Using CrowdInc to reduce costs results in an increased duration to compute final answers. Indeed, the calculus performed at round i to compute hidden variables for a task t is lost at step $i+1$ if t is not stopped. An interesting idea is to consider how a part of computations can be reused from a round to the next one to speed up convergence.

Till now, we have aggregation and a trade-off technique for a given single task. In the next chapter, we extend it to a complex workflow setting. We will see that several factors influence the realization of workflow and its cost: the number of tagging tasks that have to be realized, the available budget, the confidence in produced results, workers expertise, data, the difficulty of tagging, and the policies chosen to realize a workflow and to hire workers.

CHAPTER 6

QUALITY ASSURANCE FOR COMPLEX WORKFLOWS

6.1 Introduction

In the preceding chapters, we observed that most of the existing platforms support simple, repetitive, and independent tasks known as *micro-tasks*. They require a few minutes to an hour to complete and can be grouped into batches. However, many real-world problems are not simple micro-tasks, but rather complex orchestrations of dependent tasks, which aim are to process input data and worker's answers. The existing crowdsourcing platforms provide interfaces to execute micro tasks and access crowds but lack ways to specify and execute complex tasks. We proposed *complex workflows* (Chapter 3) which supports higher-order answers and orchestration of tasks.

Complex workflows answer the coordination of tasks but are not adapted in many crowdsourcing scenarios, especially when workers are hired to realize batches of tasks, and when redundancy is needed to guarantee data quality. Many data-centric applications come with a budget and quality constraints: as human workers are prone to errors, one has to hire several workers to aggregate a final answer with sufficient confidence. An unlimited budget would allow hiring large pools of human workers to assemble reliable answers for each micro-task, but in general, a client for a complex task imposes a limited budget B_0 that must not be exceeded. A limited budget forces us to replicate micro-tasks in an optimal way to achieve the best possible quality. The objective is then to obtain a *reliable* result, forged through a complex orchestration, and at a *reasonable cost*. In the preceding Chapter 5, we proposed the *CrowdInc* mechanism to obtain a trade-off between the quality of aggregated data and the cost to obtain an answer for a fixed batch of micro-tasks. In this chapter, we extend this technique to the complex workflow setting.

We propose a solution to address quality and cost during realization of complex tasks. We define a dataset aggregation model on top of *complex workflows*, which orchestrates tasks and distributes work according to a *dynamic policy* that considers confidence in aggregated data and the cost to increase this confidence. A workflow can be seen as an orchestration of phases, where the goal of each phase is to tag records from the dataset input to the phase. A complex task terminates when the last of its phases has completed its tagging. The output of one phase acts as an input to the next ones in the workflow. We follow the notation defined in chapter 5. For simplicity, we still consider simple Boolean tagging tasks that associate a tag in $\{0, 1\}$ to every record in a dataset. We also assume that workers are uniformly paid. We have already defined a model for compelx workflows, and a difficulty-aware adaptive aggregation mechanism. The challenge is now to execute workflows with replication of tasks, and to provide orchestration tool to realize a workflow. The overall challenge is hence to realize a workflow within budget B_0, while guaranteeing that the final dataset forged during the last phase of the workflow has a high probability to be the ground truth.

This general orchestration schema leaves several design choices open. First, the aggregation technique used influences the quality of the final results. Furthermore, the mechanisms used to decide if more workers should be hired to improve quality have an impact on costs and on the overall accuracy of answers. In the previous chapter, we saw that the simplest way to replicate micro-tasks is static execution, i.e. affect an identical fixed number of workers to each micro-task in the orchestration without exceeding budget B_0. On the other hand, one can allocate workers to tasks *dynamically* and save a large part of a budget without impacting the quality of aggregated data. This mechanism is described for a single task in Chapter 5. The principle is rather simple: hire workers for tasks with insufficient confidence if the remaining budget allows it. Improving quality and cost dynamically for workflows is more difficult and is non-trivial. Indeed, if one replicates a task in one step of a workflow, and waits for a dataset to be tagged with a sufficient confidence, he takes the risk of exhausting his budget before completion of the workflow. On the other hand, limiting the budget expenditure at each stage of a workflow amounts to static allocation of resources. We add the concept of *phase* in which workers transforms a set of records. In such cases, one can wait in each phase to achieve sufficient reliability of answers for each record of the input before forwarding data to the next phase of the workflow. This is called a *synchronous* execution of a workflow. On the other hand, one can eagerly forward records with

reliable tags to the next phases without waiting for the total completion of a phase. This is called an *asynchronous* execution.

In this work, we study execution strategies for complex workflows with different design choices. We consider several types of workflow, aggregation mechanisms (namely Majority Voting (MV) and Expectation Maximization (EM) based techniques [GC11]), several distributions of data, difficulty of tasks, workers expertise, and study the cost and accuracy of workflows on various execution strategies (static, synchronous and asynchronous execution). Unsurprisingly, the dynamic distribution of workers allow saving costs in all cases. A more surprising result is that synchronous realization of complex tasks is in general more efficient than asynchronous realization.

The rest of the chapter is organized as follows.

— Section 6.2 describes the complex workflow model with aggregation.

— Section 6.3 presents the aggregation technique used to forge final answers from the answers returned by the crowd workers. The aggregation mechanism follows from the previous chapter.

— In Section 6.4, we propose algorithms to realize complex workflows while maintaining a trade-off between cost and quality.

— We report the results of experiments on a benchmark in Section 6.5 before the conclusion.

6.2 Complex workflow with aggregation

In Chapter 3, we proposed a complex workflow model that provides a way to rewrite a complex task into a set of sub-tasks. The model provides answers for the coordination of tasks. From this model, one can check if termination and correctness are guaranteed. However, complex workflows, as introduced in Chapter 3 do not allow to consider budget and quality constraints, which are essential needs for crowdsourcing markets. The proposed *complex workflow* model is powerful, but yet we need an extension of this model that considers the aggregation technique, cost optimization, and synchronicity mechanisms. Synchronicity mechanism defines the way the task to be forwarded in the workflow. In this section, we formalize a model for *complex workflow with aggregation* (CWA). The CWA model borrows most of its ingredients from complex workflows (refer Chapter 3) but adds tasks replication, and considers aggregation and budget management in its semantics.

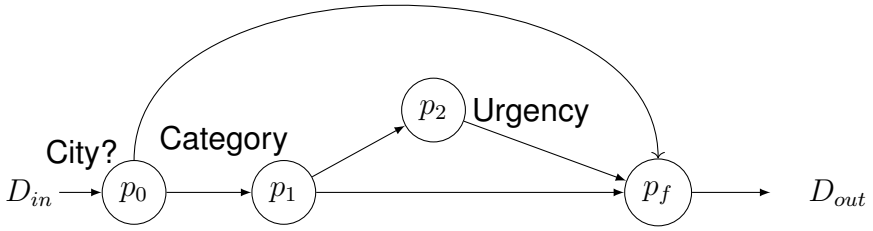


Figure 6.1 – A workflow in a smart city.

Use Case: Consider for instance online request services frequently proposed by smart cities (Figure 6.1). These systems receive a huge number of requests that must be filtered, classified, and then sent to the appropriate bureau of the city. This requires a lot of work power, but it follows a workflow made of several phases. A first step p_0 sorts a load of demands D according to the address of the client (demands from residents of the city will not be processed like external demands). The demands from city inhabitants are validated and sent to the next phase p_1. The goal of p_1 is to decide in which category (Health, Roads, ...) the request falls. After classification, phase p_2 decides whether the demand is *Urgent* or not, if some demands from that category call for rapid response. For categories that are not urgent, demands are forwarded directly to p_f. The final phase p_f gathers all annotated demands in an annotated dataset D_{out}.

Here, we can observe that the request processing is divided into several dependent phases. Each phase processes records from an input dataset or merges different inputs to a single one, and forwards the result to its successor. The CWA models follow the convention presented in Chapter 3 for data representations. We denote by FO^R the FO formulas for records, and by FO^D the FO formulas for datasets. Each phase of a CWA model consists of a tagging phase: the records input to a phase are tagged by workers and forwarded. In other words, a phase is a batch of micro-tagging tasks. For simplicity, we assume that processing a record is a micro-task that simply consists in adding a new Boolean field (called a *tag*) to this record. Hence a micro-task can be seen as an operation that transforms a record $r(v_1, \ldots, v_k)$ into a new record $r'(v_1, \ldots, v_k, v_{k+1})$ where v_{k+1} is a Boolean value. The setting can be easily adapted to let v_{k+1} take values from a discrete domain. In the previous Chapter 5, we saw that humans are prone to errors, micro-tasks are replicated and allocated to several workers. CWA keeps this philosophy: the tagging operation in each phase might be incorrectly performed by a single worker, so each micro-task associated with a given phase needs to be replicated. Then, it requires an aggregation mechanism to combine answers be-

fore proceeding to the next phases. We showed a simple data forwarding mechanism in the *Complex Workflow* model, where the only requirement to start a task is that none of its input is empty. For CWAs, we implement a more complex flow for data: we allow to distinguish paths followed by records from a dataset depending on the values attached to their fields. The result of each phase must be forwarded to its successors. As shown by the example in Figure 6.1, the contents of records must be used to decide whether a record in a dataset has to be forwarded to a successor phase or not. Hence, phases are used to tag records, aggregate answers, but also play the role of filters. When a phase has several successors, the contents of records are used to decide to which successor(s) it should be forwarded to. This allows to split datasets according to the value of a particular data field, process different records depending on their contents, create concurrent threads, etc. We define *Complex Workflow with Aggregation* as follows.

Definition 33 (Complex Workflow with Aggregation)**.** *A complex workflow with Aggregation is a tuple* $CWA = (\mathcal{P}, \longrightarrow, Ag, G, \bigotimes, p_0, p_f)$ *where* $\mathcal{P}$ *is a finite set of phases,* p_0 *is a particular phase without predecessor,* p_f *a phase without successor,* $\longrightarrow \subseteq \mathcal{P} \times FO^R \times \mathcal{P}$ *is a flow relation and* $G = \mathcal{P} \rightarrow FO^D$ *associates a guard to every phase, and for every* $p_x \in \mathcal{P}$*,* $\bigotimes(p_x)$ *is an operator used to merge input datasets.*

Intuitively, a phase associates a tagging task to each record in its input dataset, replicates and distributes it to several workers. The answers returned by all the workers are then aggregated to get a final trusted answer. As in Chapter 5, we assume that worker's answers are independent. For a triple $(p_x, g_{x,y}, p_y)$ in $\longrightarrow$, we will say that p_x is a predecessor of p_y and p_y is a successor of p_x. We denote by $Succ(p_x) = \{p_y \mid p_x \longrightarrow^* p_y\}$ the set of phases that must occur after p_x and by $Pred(p_x) = \{p_y \mid p_y \longrightarrow^* p_x\}$ the set of phases that must occur before p_x. The meaning of guard $g_{x,y}$ is that every record produced by phase p_x that satisfies guard $g_{x,y}$ is forwarded to p_y. We will see in the rest of this section that "producing a record" is not done in a single shot, and requires duplicating a tagging micro-task, aggregate answers, and decide if the confidence in the aggregated answer is sufficient.

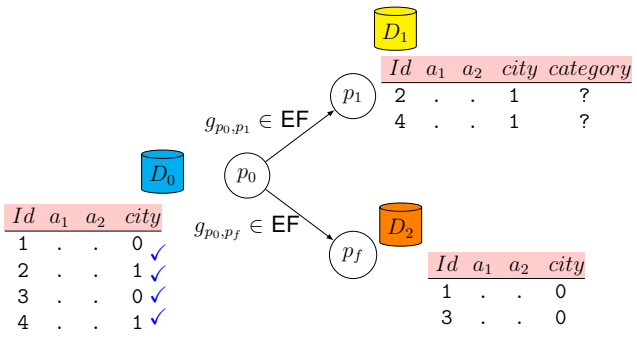


Figure 6.2 – Exclusive fork.

When a phase p_x has several successors $p_y^1, \ldots p_y^k$ and the guards $g_{x,y}^1, \ldots g_{x,y}^k$ are exclusive, each record processed by p_x is sent to at most one successor. We will say that p_x is an *exclusive fork* (EF) phase.

Example (Exclusive Fork). Consider the example of the exclusive fork in Figure 6.2(a snippet of the smart city workflow). The phase p_0 is attached with dataset $D_0 = rn(Id, a_1, a_2, city)$ and has successor phases p_1 and p_f. The checkmark ✓ indicates that worker has processed the record and $?$ indicates that workers are still executing and working on the record. We follow this convention throughout the chapter. Here, all records in dataset D_0 have been tagged by workers and the aggregated answer has a sufficient confidence. The tagging task of phase p_0 is hence finished. Let g_{p_0,p_1} and g_{p_0,p_f} be exclusive guards of the form $g_{p_0,p_1} = (\text{"}city = 1\text{"})$ and $g_{p_0,p_f} = (\text{"}city = 0\text{"})$. This allows to forward the records from the phase p_0 to at most one phase. Guard g_{p_0,p_1} states that all records tagged with $city$ value 1 is to be forwarded to phase p_1 and the records with $city$ value 0 is to be sent to phase p_f. Hence, on the example dataset D_0 of Figure 6.4, the records with id value equals 2 and 4 are forwarded to the phase p_1 as dataset D_1, whereas the records with id value 1 and 3 are forwarded to phase p_f in dataset D_2.

On the contrary, when guards are not exclusive, a copy of each record processed in p_x can be sent to each successor (hence increasing the size of data processed in the workflow), and p_x is called a *non-exclusive fork* (NEF) phase.

Example (Non-exclusive Fork). We give an example in Figure 6.3 to illustrate the non-exclusive forks. Similar to the above example, the phase p_0 is tagged with dataset $D_0 = rn(Id, a_1, a_2, city)$ and has successor phases as p_1 and p_f. Let g_{p_0,p_1} and g_{p_0,p_f} be non-exclusive guard. With these non exclusive guards, a copy of record can be

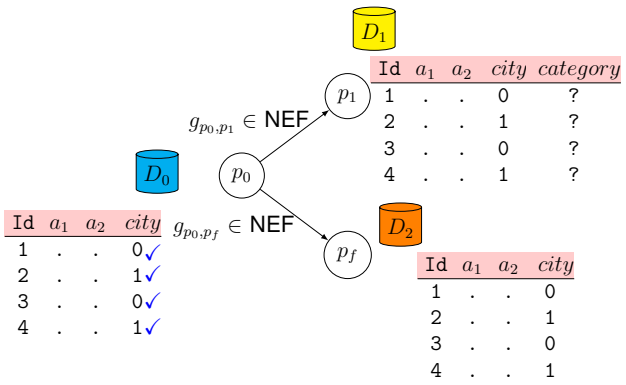


Figure 6.3 – Non-exclusive fork.

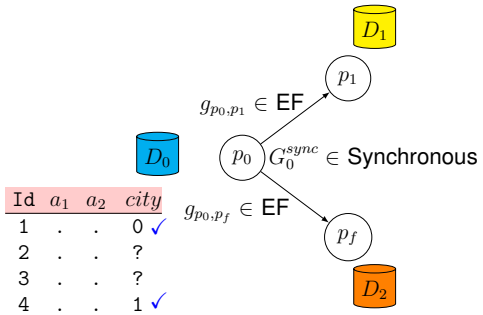


Figure 6.4 – Synchronous execution.

forwarded both to p_1 and to p_f. In our example, the non exclusive fork replicates all records in dataset D_1 (used as an input for p_1) and D_2 used as an input for p_f.

For a given phase $p_x \in \mathcal{P}$, we denote by $G_x \in FO^D$ the guard attached to phase p_x. G_x addresses properties of the dataset input to p_x by its predecessors. This allows, in particular, to require that all records in preceding phases have been processed (we will then say that phase p_x is *synchronous*), that at least one record exists in some predecessor (the task is then fully *asynchronous*), or any FO expressible property on datasets produced by predecessors of p_x.

Example (Synchronous and Asynchronous execution). One can easily define guards for phases that force synchronous and asynchronous execution of phases. Consider for instance the fragment of the smart city example in Figure 6.1. Let D_0 denote the dataset produced by phases p_0. The guard $G_0^{sync} := \forall rn(id, a_1, a_2, city) \in D_0, city \neq ?$ imposes that phase p_1 starts only when all records have been processed by p_0. On the example of Figure 6.4, the blue checkmark indicates that records 1, 4 have been processed. However, records 2 and 3 are still being processed by workers. In a similar way, one can

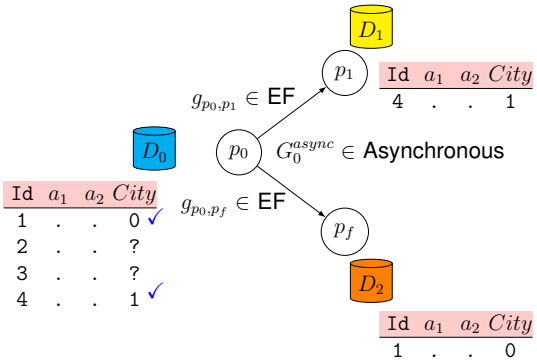


Figure 6.5 – Asynchronous example.

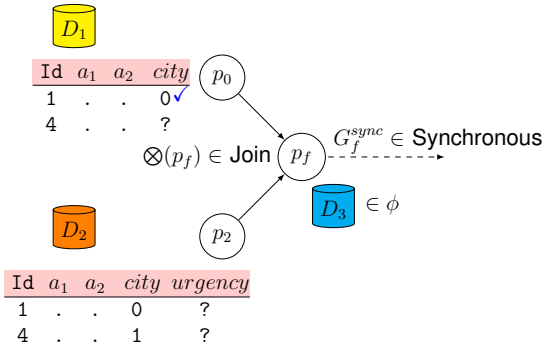


Figure 6.6 – Join example.

define an asynchronous guards of the form $G_0^{async} := \exists rn(id, a_1, a_2, city) \in D_0, city \neq ?$ that indicates processing of start in p_1 can start as soon as record have been processed by p_0 and is shown in Figure 6.5.

One can notice that a phase p_x (e.g. p_f in the example of Figure 6.1) may have several processors. In this case, we consider that the input of p_x is obtained by merging all datasets produced by its predecessors. We denote by $\bigotimes_x = \bigotimes(p_x)$ the operator used to merge all inputs entering phase p_x. This operator can be either a simple union of datasets, or a more complex join operation. If $\bigotimes_x$ is a join operation, we impose that p_x is synchronous. This is mandatory, as one cannot start processing data produced by a join operation when the final set of records is not known.

Example (Join operation). We show the Join operation in Figure 6.6 from our previous smart city example. Phase p_f takes the input from the predecessor phases p_0, p_2 and performs *join* operation. Note that, phase p_f with guard $G_f^{sync} \in$ *Synchronous*, waits for all the records from the predecessor phase to get processed. Here in the Figure 6.6,

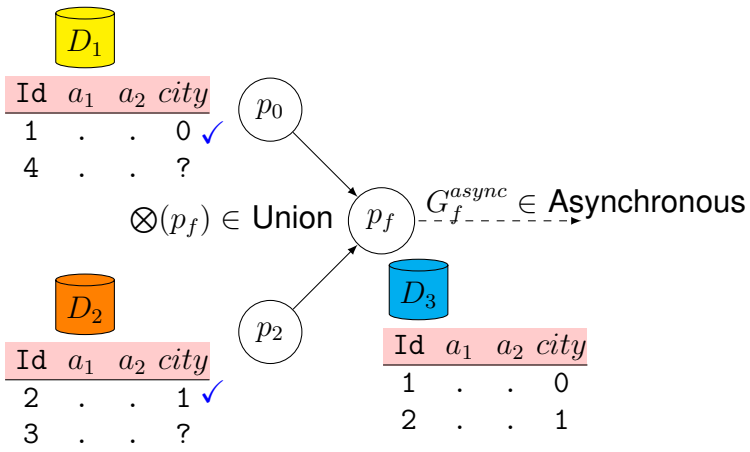


Figure 6.7 – Union example.

records with id value 1 in phase p_0 has been processed and the rest of the records are still being executed by workers in phase p_0 as well as in p_2. Consequently, phase p_f is waiting for all the data and hence D_3 is empty.

When $\bigotimes_x$ is a simple union of datasets, as tasks are individual tagging of records, any record processed on a predecessor of p_x can be processed individually without waiting for other results to be available. This allows asynchronous executions in which two phases p_x, p_y can be concurrently active (i.e. have started processing records), even if p_x precedes p_y.

Example (Union operation). An example of *union* operation is shown in Figure 6.7. Here a phase p_f is attached with a *union* operation with inputs as dataset D_1, D_2 and the guard $G_f^{async} \in$ *Asynchronous*. Union operation simply stacks up the records and the records can be processed individually. Hence, the processed record with id value 1 from dataset D_1 and the record with id value 2 attached to dataset D_2 is forwarded to phase p_f which performs the union operation and we get the dataset D_3. Notice however that due to asynchronous realization of p_f, D_3 will be processed as soon as a record arrives at phase p_f.

The semantics of CWA is defined in terms of moves from a configuration to the next one, organized in *rounds*. Configurations memorize the data received by phases, a remaining budget, the answers of workers, and quality measures on answers and workers competences.

Definition 34. *A* configuration *of a CWA is a tuple* $C = (\mathcal{D}_{in}, W_{in}, W_{out}, conf, B_r)$ *where*

- $\mathcal{D}_{in} : \mathcal{P} \rightarrow Dsets$ *associates a dataset to every phase* $p_x \in \mathcal{P}$ *(this dataset can be empty).*
- $W_{in} : \mathcal{P} \times \mathbb{N} \times \rightarrow 2^W$ *associates a set of workers to each record* r_j *in* $\mathcal{D}_{in}(p_x)$*.*
- $W_{out} : \mathcal{P} \times \mathbb{N} \times W \rightarrow \{0, 1\} \cup \emptyset$ *associates a tag or the empty set to a worker, a phase and a record number that was formerly affected to the worker. We will write* $l^x_{i,j}$ *to denote the answer returned by worker* w_i *when tagging record* r_j *during phase* p_x*.*
- $conf : \mathcal{P} \times \mathbb{N} \rightarrow [0, 1]$ *is a map that associates to each record* r_j *in* $\mathcal{D}_{in}(p_x)$ *a confidence score in* $[0, 1]$ *computed from* W_{out}
- $B_r \in \mathbb{N}$ *is the remaining budget.*

$W_{out}(p_x, j, i) = \emptyset$ indicates that a worker w_i in a phase p_x has not yet processed record r_j. We say that phase p_x is *completed* for a record r_j from $\mathcal{D}_{in}(p_x)$ if there is no worker w_i such that $W_{out}(p_x, j, i) = \emptyset$ and $w \in W_{in}(p_x, r_j)$. As soon as phase p_x is completed for r_j, we can derive an *aggregated answer* $r'_j(v_1, \ldots, v_n, y^x_j)$ for each record $r_j(v_1, \ldots v_n)$ from the set of all answers returned by the workers in $W_{in}(p_x, j)$. Similarly, we can compute a confidence score $conf(p_x, j)$ on the value y^x_j and update the expertise of each worker (we will see how these values are evaluated in Section 6.4). In a configuration, phases and records can be in three states: inactive, active and closed. We say that a phase p_x is *inactive* if it has no copies of record assigned and a record r_j is *inactive* in a phase p_x if no workers is assigned to it till now. A phase p_x is *active* if some record r_j is being processed and a record r_j is *active* in a phase p_x if it is being processed by some workers. Given a threshold value Th, we will say that p_x is *closed* for a record r_j from $\mathcal{D}_{in}(p_x)$ if $conf(p_x, j) \geq Th$. Record $r'_j(v_1, \ldots, v_n, y^x_j)$ will then be part of the input of phase p_y if $(p_x, g_{x,y}, p_y) \in \longrightarrow$ and $r'_j(v_1, \ldots, v_n, y^x_j)$ satisfies guard $g_{x,y}$. We will say that phase p_x is closed if all its predecessors are closed, and for every $j, r_j \in \mathcal{D}_{in}(p_x), conf(p_x, j) \geq Th$. A *closed* phase has no records left to process and all the records in the preceding phases are also closed.

We can now detail how rounds change the configuration of a workflow. The key idea is that each round aggregates available answers, and then decides whether the confidence in aggregated results is sufficient. If confidence in a record is high enough, this record is forwarded to the successor phases, if not new workers are hired for the next round, which decreases the remaining budget. The threshold for the confidence decreases accordingly. Then new workers are hired for freshly forwarded data, leaving

the system ready for the next round. From a configuration $C = (\mathcal{D}_{in}, W_{in}, W_{out}, conf, B)$, a round produces a new configuration $C' = (\mathcal{D}_{in}, W_{in}, W_{out}, conf, B)$ as follows:

— **Answers**: Workers that were hired in preceding round produce new data. For every phase p_x every record $r_j \in \mathcal{D}_{in}(p_x)$ and every worker w_i such that $w_i \in W_{in}(p_x, j)$ and $W_{out}(p_x, j, i) = \emptyset$, workers w_i produce a new output $l^x_{i,j} \in \{0, 1\}$ and we hence set $W_{out}(p_x, j, i) = l^x_{i,j}$ [1].

— **Aggregation**: The system evaluates aggregated answers in every active phase p_x. For every record r_j in $\mathcal{D}_{in}(p_x)$, we compute an aggregated answer y^x_j from the set of answers $A_j = \{l^x_{i,j} \mid w_i \in W_{in}(p_x, j)\}$. We derive tasks difficulty and workers expertise and also compute a new confidence score $conf'(p_x, j)$ for the aggregated answer (this confidence depends on the aggregation technique shown in Section 5.2).

— **Data forwarding** : We distinguish asynchronous and synchronous phases. Let p_y be an asynchronous phase ($\otimes_y$ can only be a union of records). Then p_y accept every new record $r'(v_1, \ldots v_n, y^x_j)$ that was not yet among its inputs from a predecessor p_x provided r' satisfies guard $g_{x,y}$, and the confidence in the aggregated answer y^x_j is high enough. Formally, $\mathcal{D}'_{in}(p_y) = \mathcal{D}_{in}(p_y) \cup \{r'(v_1, \ldots v_n, y^x_j)\}$ if $(p_x, g_{x,y}, p_y) \in \longrightarrow$, $conf'(p_x, j) \geq Th$ and $r'(v_1, \ldots v_n, y^x_j) \models g_{x,y}$.
Let p_y be a phase such that $\otimes_y$ is synchronous. If there exists a predecessor p_x of p_y that is not *closed*, then $\mathcal{D}'_{in}(p_y) = \emptyset$ (shown in Figure 6.6). Otherwise we can compute an input for phase p_y as a join over datasets computed by all preceding phases. Formally,

$$\mathcal{D}'_{in}(p_y) = \bigotimes_{\substack{y \\ p_x \longrightarrow p_y}} D_x$$

where p_x ranges over the set of predecessors of p_y, and

$$D_x = \{ \quad r(v_1, \ldots v_n, y^x_j) \mid r(v_1, \ldots v_n) \in \mathcal{D}_{in}(p_x) \wedge r(v_1, \ldots v_n, y^x_j) \models g_{x,y}\}$$

Hence, for synchronous a phase p_y, the input dataset is a join operation computed over datasets filtered by guards and realized only once predecessor tasks have produced all their results with a high enough confidence score. Both in

1. The value will be sampled according to worker's expertise and ground truth during our experiments in Section 6.5. In a real non-simulated crowdsourcing setting, answers are the tag obtained by assigning task to workers registered on the platform.

synchronous and asynchronous settings, phase p_y becomes *active* if $\mathcal{D}'_{in}(p_y) \models G(p_y)$, and we set $conf'(p_y, j) = 0$ for every new record in $\mathcal{D}'_{in}(p_y)$

— **Worker allocation** : For every p_x that is *active* and every record $r_j = r(v_1, \ldots v_n) \in \mathcal{D}'_{in}(p_x)$ such that $conf'(p_x, j) < Th$, we allocate k new workers $w_1, \ldots w_k$ to record r_j for phase p_x, i.e. $W'_{in}(p_x, j) = W_{in}(p_x, j) \cup \{w_1, \ldots, w_k\}$. The number of workers depend on the chosen policy (see details in Section 6.4). Accordingly, for every new worker w_i affected to a tagging task for a record r_j in phase p_x, we set $W'_{out}(p_x, i, j) = \emptyset$

— **Budget update**. We then update the budget. The overall number of workers hired is

$$jw = \sum_{p_x \in \mathcal{P}} \sum_{r_j \in \mathcal{D}'_{in}(p_x)} |W'_{in}(p_x, j) \setminus W_{in}(p_x, j)|$$

We consider, for simplicity, that all workers and tasks have identical costs, we hence set $B'_r = B_r - jw$.

An execution begins from an initial configuration C_0 in which only p_0 is active, with an input dataset affected to p_0, and starts with workers allocation. Executions end in a configuration C_f where all records in $\mathcal{D}_{in}(p_f)$ are tagged with a sufficient threshold, or in a configuration where the remaining budget is 0. Notice that several factors influence the overall execution of a workflow. First of all, the way workers answers are aggregated influences the number of workers that must be hired to achieve decent confidence in the synthesized answer. We propose to consider two main aggregation policies. The first one is majority voting (MV), where a fixed static number of workers is hired for each record in each phase. A second policy is the expectation maximization (EM) based technique already used in Chapter 5, in which workers are hired on-demand to increase confidence in the aggregated answer. With this policy, the confidence level is computed taking into account the estimated expertise of workers, and the difficulty of records tagging. The number of workers hired per record in a phase is not fixed but rather computed considering the difficulty of tagging records and the remaining budget.

For a given workflow, asynchrony is another key factor that may influence the accuracy and budget spent to realize a complex task. Recall that for a phase p_x which input dataset is built as a union of sets of records, asynchronous guards allow to start processing records as soon as $\mathcal{D}_{in}(p_x) \neq \emptyset$. Synchronous guards force p_x to wait for the termination of its predecessors. In Section 6.5, we study the impact of synchronous/ asynchronous execution policy on the overall cost and accuracy of a workflow.

6.3 Aggregation Model

We use the aggregation mechanisms for boolean tasks shown in Chapter 5. As mentioned earlier, crowdsourcing requires replication of micro-tasks, and aggregation mechanisms for the answers returned by the crowd.

Consider a phase p_x which input is a set of records $D_x = \{r_1, r_2, \ldots, r_n\}$, and which goal is to associate a Boolean tag to each record of D_x. We assume a set of k independent and faithful workers that return Boolean answers. We denote by l_{ij} the answer returned by worker j for a record r_i. $L_i = \bigcup_{j \in 1 \ldots k} l_{ij}$ denotes the set of answers returned by k workers for a record r_i and $L = \bigcup_{j \in 1..n} L_j$ denotes the set of all answers and the the objective of aggregation is to derive a set of *final answers* $Y = \{y_j, 1 \leq j \leq n\}$ from the set of answers L. Once a final answer y_j is computed, it can be appended as a new field to record r_j. As explained in Section 6.2, the produced results can be used to launch new phases and to forward records to successor phases of p_x.

We recall the notation used in Chapter 5: $d_j \in [0, 1]$ denotes the difficulty to tag a record r_j. The expertise of a worker is denoted as a pair $\xi_i = \{\alpha_i, \beta_i\}$, where α_i is the **recall** and β_i the **specificity** of worker i. We use the same generative model as in Chapter 5, i.e. $Pr(l_{ij} = y_j | d_j, \alpha_i, y_j = 1) = (1 + (1 - d_j)^{(1-\alpha_i)})/2$ and $Pr(l_{ij} = y_j | d_j, \beta_i, y_j = 0) = (1 + (1 - d_j)^{(1-\beta_i)})/2$ to model how the worker answers. We use both majority voting to aggregate answers or as in Chapter 5, a more clever EM based algorithm that estimates difficulty of task d_j and expertise of workers α_i, β_i and returns the final aggregated answer y_j for each of the record.

6.4 Cost Model for Workflow

The objective of a complex workflow W over a set of phases $P = \{p_0, \ldots, p_f\}$ is to transform a dataset input to the initial phase p_0 and eventually produce an output dataset. The final answer is the result of the last processed phase p_f. The simplest scenario is a workflow that adds several binary tags to input records. The realization of a micro-task by a worker is paid, and workflows come with a fixed maximal budget B_0 provided by the client. For simplicity, we consider that each worker receives one unit of credit per realized task. As explained in Section 6.2, each phase receives records and each record is tagged by one or several workers. Answers are then aggregated,

and the records produced by a phase p_x are distributed to its successors if they meet some conditions on the data. A consequence of this filtering done by conditions is that records have different lifetimes and follow different paths in the workflow. Further, one can hire more workers to increase confidence in an aggregated result if needed and if a sufficient budget remains available. Several factors influence the realization of a workflow and its cost: the number of tagging tasks that have to be realized, the available initial budget, the confidence in produced results, workers expertise, the size and nature of input data, the difficulty of tagging, and the policies chosen to realize a workflow and to hire workers. As for the single task case considered in Chapter 5, static allocation of workers to tag each record in each phase is not optimal, as the same effort is spent to solve easy and difficult tasks. Further, if the workflow execution spends most of the budget on easy records or on the short branches of the workflow it may reduce the accuracy of the workflow. The static allocation of workers also does not allow to save budget on easy tasks and reuse these resources to improve the accuracy of the results.

In Section 6.2, we defined synchronous and asynchronous schemes to allocate workers on-the-fly to tasks. The decision to hire more workers for a particular record r_j processed in a phase depends on the confidence in the aggregated result, and on the threshold. In this section, we define the cost model associated with these schemes, and in particular, the threshold measure used to decide whether more workers should be hired. We show in Section 6.5 that the algorithm achieves a good trade-off between cost and accuracy. Recall that at each round, we allocate new micro-tagging tasks to workers, to obtain answers for records that are still open. EM aggregation is used to obtain a plausible aggregated tag y_j^x for each record r_j from the set of answers L_j^x obtained in each active phase p_x. The algorithm also estimates difficulty d_j^x of tagging record $r_j \in p_x$, and evaluates the expertise level of every worker w_i, i.e. its recall α_i and its specificity β_i. These values are used to compute a *confidence score* $\hat{c}_j^x$ for the aggregated answer. This score is used to decide whether more answers are required or y_j^x can be considered as a definitive result. Let $k_j^x = |L_j^x|$ denote the number of answers for record $r_j \in p_x$. The *confidence* $\hat{c}_j^x$ in the aggregated answer y_j^x is defined as:

$$\hat{c}_j^x(y_j^x{=}1) = \frac{1}{k_j^x} \cdot \sum_{i=1}^{k_j^x} \left\{ l_{ij}^x \times \left(\frac{1+(1-d_j^x)^{(1-\alpha_i)}}{2}\right) + (1 - l_{ij}^x) \times \left(1 - \frac{1+(1-d_j^x)^{(1-\alpha_i)}}{2}\right) \right\}$$

$$\hat{c}_j^x(y_j^x{=}0) = \frac{1}{k_j^x} \cdot \sum_{i=1}^{k_j^x} \left\{ (1 - l_{ij}^x) \times \left(\frac{1+(1-d_j^x)^{(1-\beta_i)}}{2}\right) + (l_{ij}^x) \times \left(1 - \frac{1+(1-d_j^x)^{(1-\beta_i)}}{2}\right) \right\}$$

One can notice that we reuse the confidence scores proposed in Chapter 5. Confidence $\hat{c}_j^x$ depends on the *recall* and *specificity* of workers providing an answer to tag record r_j. This value is then compared to threshold Th. If $\hat{c}_j^x$ is greater than a current threshold Th, then answer y_j^x is considered as definitive and the record r_j is closed. Otherwise, the record remains active. We fix a maximal number $\tau \geq 1$ of workers that can be hired during a round for a particular record. Let T_{ar} denote the set of active records after aggregation and D_{max}^x the maximal difficulty for a record of phase p_x in T_{ar}. For every record $r_j^x \in T_{ar}$ with difficulty d_j^x, we allocate $\mathbf{a}_j^x = \lceil (d_j^x / D_{max}^x) \times \tau \rceil$ new workers for the next round. Intuitively, we allocate more workers to more difficult tasks. Now, T_{ar} and hence $\mathbf{a}_j^x$ depend on the threshold computed at each round. An appropriate threshold must consider the remaining budget, the remaining work to do, that depends on the number of records that still need to be processed, on the structure of the workflow, and the chosen policy to orchestrate the workflow and transfer data from phase to phase. A first parameter to fix for the realization of a workflow is the initial budget of B_0. Ideally, a minimal budget B_0 should allow the realization of a workflow, i.e. provide sufficient resources to hire workers at each phase for each record. Henceforth, the budget must consider the structure of the workflow.

Definition 35 (Height, width)**.** *The* height *of phase $p_x \in \mathcal{P}$ is the length of the longest path from p_x to the final aggregation phase p_f and is denoted by h_x. The height a workflow W is the height of its initial phase p_0 and is denoted by h_0. The* width *of W is the size of the largest clique in W (i.e. the largest subset $X \subseteq \mathcal{P}$ such that $\forall x, y \in X, x \not\rightarrow y$).*

The notions of height and width are interesting to evaluate the required budget to terminate execution of a workflow, and hence fix an appropriate threshold for confidence. The height of a workflow represents the maximal number of data transmission

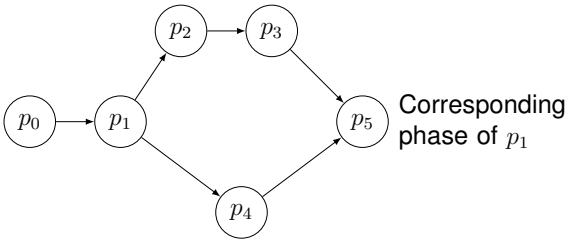


Figure 6.8 – Corresponding phase example.

from one phase to the next one needed to move a record from p_x to p_f. The width of a workflow is bound on a maximal number of active copies or a record at a given instant. Before considering how the threshold is computed, we need to verify that a workflow has a sufficient budget for its execution. In each phase, a record is allocated at least $\zeta \geq 1$ workers. So, if n is the total number of records to process in a workflow W, the budget spent during the realization of W will lay between $\zeta.n.h_0$ and B_0. Similarly, some records are sent to more than one successor, so in a workflow of width w, we may have to fund up to $\zeta.n.h_0.w$ micro-tasks to terminate. However, this is a very coarse approximation of the minimal resources needed to terminate a workflow, and a micro-task may stay for more than one record in a phase.

To obtain sharper evaluations, we first compute a bound on the number of remaining phases that records have to go through when they are currently processed in a phase p_x before completion of the workflow. We call this number the *foreseeable work* at phase p_x and denote it by $fw(p_x)$. For readability, we assume simple non-hierarchical structures, i.e. without nesting of exclusive and non-exclusive forks. For these structures, this bound can be computed iteratively according to the structure of the workflow. A similar bound can be computed inductively for more complex workflows with nested exclusive/non-exclusive forks.

Let p_x be a phase with several successors. If p_x is a non-exclusive fork phase, then there is no other fork along a branch before a merge phase. A phase p_y that is a merge phase immediately after p_x is called the *corresponding phase* of p_x. As for example, in the Figure 6.8 p_1 is a non-exclusive fork and the corresponding phase of p_1 is p_5. Intuitively, in a realization of W, all phases on a path between p_x and p_y can only process records that went through p_x. Hence, the workload to process a record from p_x to p_y is exactly the size of $Succ(p_x) \cap Pred(p_y)$.

Let p_x be an exclusive fork phase. Then, the corresponding union node of p_x is a node p_y such that all path originating from p_x visit p_y, and no predecessor of p_y satisfy

this property. To decide whether p_y is the corresponding node of p_x one can compute the set of nodes that are both successors of p_x and predecessor of p_y, and decide for each of them whether they have a successor that is not a predecessor of p_y. This can be done in polynomial time. Then, the maximal number of phases that have to be realized between p_x and its corresponding node p_y is the length of the maximal path from p_x to p_y (which can also be computed in PTIME [Kha11]).

The algorithm to compute the foreseeable workload in a phase p_x consists in computing the most expensive path in a workflow once the cost of parallel processes contained between a non-exclusive fork node and its corresponding phase is evaluated and this set of nodes is replaced by a single phase of the corresponding cost. This is described more precisely in Algorithm 3.

Algorithm 3: $FW(p_n)$

Data: Workflow W, phase p_n
Result: $fw(p_n)$
for *each node* p_x *in* $Succ(p_n)$ **do**
  $Cost(p_x) := 1$;
end
for *each non-exclusive fork node* p_x *in* $Succ(p_n)$ **do**
  Find p_y the corresponding node of p_x ;
  Z = $Succ(p_x) \cap pre(p_y)$;
  Replace Z by a fresh node p'_x ;
  $Cost(p'_x) = |Z|$;
end
Compute the most expensive path $p_n.p_{i_1}. \ldots .p_{i_k}.p_f$ from p_n to p_f;
$fw := 1 + \sum Cost(p_{i_k})$;
return fw;

Definition 36 (Foreseeable workload)**.** *Let n_x denote the total number of active records at phase p_x in a configuration C. The foreseeable task number from p_x in configuration C is denoted $ft_C(p_x)$ and is defined as $ft_C(p_x) = n_x \times FW(p_x)$. The foreseeable task number in configuration C is the sum $FTN(C) = \sum_{p_x \in \mathcal{P}} ft_C(p_x)$*

Example: We illustrate the calculus of foreseeable workload with an example. Consider the workflow W with finite set of phases $\mathcal{P} = \{p_0, p_1, p_2, p_3, p_4, p_5, p_f\}$ shown in Figure 6.9. The flow relation is defined as $p_0 \longrightarrow p_1, p_0 \longrightarrow p_f, p_1 \longrightarrow p_2, p_1 \longrightarrow p_3, p_2 \longrightarrow p_4, p3 \longrightarrow p_4, p_4 \longrightarrow p_5, p_4 \longrightarrow p_6, p_5 \longrightarrow p_f, p_6 \longrightarrow p_f$. Consider the phase

p_1 is a non-exclusive fork and the rest of them are exclusive ones. Note that, when a phase p_i is non-exclusive fork, each record processed in p_i can be forwarded to each successor and hence increases the size of data processed in the workflow. Let us compute the foreseeable workload for phase p_0. We first set the cost of each phase p_x to 1, $Cost(p_x) := 1$. Then for the non-exclusive fork p_1, we find the *corresponding phase* of p_1 which is p_4. The phases between the non-exclusive fork in the dashed area in Figure 6.9 and corresponding phase, p_2 and p_4 are replaced by phase p'_1 to obtain a workflow W' in Figure 6.10. The cost of the phase p'_1 is $|\{p_2, p_3\}|$ and hence $Cost(p'_1) = 2$. Now, let us compute the foreseeable workload for phase p_0 in W'. The most expensive path from p_0 is $p_0.p_1.p'_1.p_4.p_5$. Hence $fw(p_0) = Cost(p_0) + Cost(p_1) + Cost(p'_1) + Cost(p_4) + Cost(p_5) = 6$. The same algorithm can be used to derive the foreseeable workload for the rest of the phases in W.

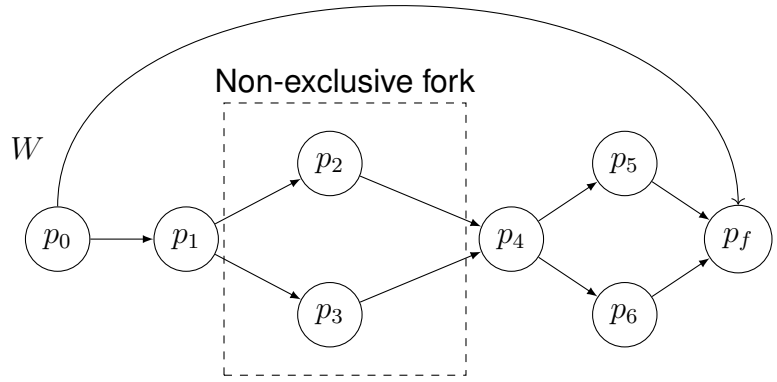


Figure 6.9 – Workflow W with non-exclusive fork phase.

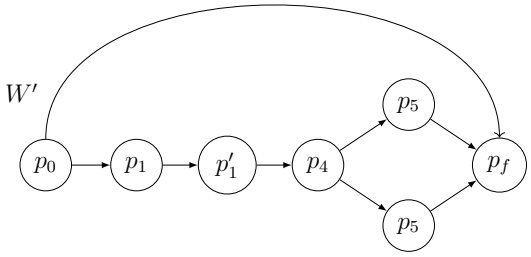


Figure 6.10 – Workflow W' with a new phase p'_x replacing the phases p_2, p_3 of the workflow W.

We are now ready to define a threshold function based on the current configuration of the workflow. This function must consider all records that still need processing, the remaining budget, and an upper bound on the number of tagging tasks that will have to be realized to complete the workflow. Further, the execution policy will influence the way workers are hired, and hence the budget spent. In a synchronous execution, records in a phase p_x can be processed only when all records in preceding phases have been processed. On the contrary, in asynchronous execution mode, processing of records input to a phase p_x can start without waiting for the closure of all records input to preceding phases. A consequence is that in synchronous modes, decisions can be taken *locally* to each phase, while in an asynchronous mode, the way to tune threshold must be taken according to a *global* view of the remaining work in the workflow. Hence, for an asynchronous execution policy, we will consider a global threshold function, computed for the whole workflow. On the contrary, for a synchronous execution policy, we will define a more local threshold function computed for each phase.

Asynchronous execution. We define the global ratio $\Gamma \in [0, 1]$ of executed work in a workflow. The execution of a workflow starts from a configuration C_0 with an expected workload of $FTN(C_0)$. It is an upper bound, as all records do not necessarily visit the maximal number of phases. Indeed, the phases visited by a record may depend on the answers returned by the workers. We define a ratio depicting the proportion of already executed or avoided work in a configuration C as $\Gamma = \frac{(FTN(C_0)-FTN(C))}{FTN(C_0)}$. Note that at the beginning of execution, $\Gamma = 0$ as no record is processed yet. When records are processed and moved to successor phases, Γ increases, and we necessarily have $\Gamma = 1.0$ when no record remains to process. Now, the threshold value has to account for the remaining budget to force the progress of records processing. Let B_0 denote the budget at the beginning of execution that is provided by the client, and B^c be the budget consumed. We denote by δ the fraction of budget consumed at a given execution time, i.e. $\delta = \frac{B^c}{B_0}$. Note that, at the beginning of an execution, $B^c = 0$ and hence $\delta = 0$. δ increases at every round of the execution, and takes value $\delta = 1$ when the whole budget is spent. We now define a global threshold value $Th \in [0.5, 1.0]$ that accounts for the remaining work and budget.

$$Th = \frac{1 + (1 - \delta)^{\Gamma}}{2} \tag{6.1}$$

We remind that in a phase p_x, a record which aggregated answer y_j^x has a confidence level $\hat{c}_j \geq Th$ is considered as processed for phase p_x. In an asynchronous execution policy, the threshold is a global value and applies to all records in the workflow at a given instant. The intuition for Th is simple: when only a few records remain to be processed, and the remaining budget is sufficiently high, then one can afford hiring more answers to get final answers with a high confidence threshold. This means that many records will obtain new answers and probably increase their current confidence level. Conversely, if the number of records to be processed is high and the remaining budget is low, then the threshold decreases, and even records which current answer have a low confidence level are considered as processed and moved to the next phase(s).

Synchronous execution. In asynchronous execution, records are processed individually, and a phase does not wait for the completion of its predecessors to start. As a consequence, all phases of the workflow can be active at the same time. As in each round, records can be processed in all phases, we consider a global threshold, and hence a global budget allocation policy. However, in synchronous executions, records are processed phase by phase, i.e. a phase does not start processing its input dataset until all records in the preceding phases have been processed. Here, the global threshold used in asynchronous executions is not appropriate for synchronous executions. As tasks are realized phase by phase, in the early phases of workflows execution, the threshold will be high. One may face situations where at the end of an execution, the number of records to process is still high, but a significant part of the budget has already been used. Within this setting, at the end of the execution the confidence and threshold is low. It forces to accept final answers with low confidence and hence possibly wrong answers. This may affect the overall quality of the final output of the workflow. To avoid this problem, we propose to allocate the budget phase by phase. The idea is to divide the budget among phases based on the number of records processed using the threshold per active phase.

In synchronous executions, phases start processing all their records once preceding phases have completed their work (i.e. all records are processed and for each of them, a final answer with a sufficient confidence score has been aggregated). A task becomes *active* when it starts processing records. We denote by $Init(p_x)$ the number of records input to p_x when the phase becomes active.

As for an asynchronous execution, a synchronous execution starts with an initial budget of B_0. The algorithm works in rounds and at each round, a part of the remaining budget is spent to hire the workers. As for asynchronous executions, we denote by B^c the consumed budget in a given configuration and the remaining budget $\Delta = B_0 - B^c$. The key idea in synchronous execution is to compute resources needed for each active phase, a ratio of input records that still need additional answers to forge a trusted answer, and a local threshold.

The initial budget allocated to a phase p_x when it becomes active to execute $Init(p_x) = n_x$ number of records in a configuration C is

$$B_x^{in} = \frac{\Delta}{\sum_{p_i \in \text{ Active Phases }} FTN(p_i)} \times n_x$$

Intuitively, one shares the remaining budget among active phases to allow termination of the workflow from each phase. Then for each active phase p_x, we maintain the consumed budget B_x^c, and $\delta_x = \frac{B_x^c}{B_x^{in}}$ denotes the ratio of the consumed budget in phase p_x with respect to initial allocated budget B_x^{in}. Now, for each active phase, we compute the ratio Γ_x depicting the proportion of already closed records in comparison to the total initial records attached to the phase p_x as

$$\Gamma_x = \frac{|\{r_j \mid \hat{c}_j \geq Th_x\}|}{Init(p_x)} \tag{6.2}$$

where Th_x is the threshold computed for the previous round. A local threshold for the realization of the next round of an active phase can then be computed as in asynchronous execution, using the formula

$$Th_x = \frac{1 + (1 - \delta_x)^{\Gamma_x}}{2} \tag{6.3}$$

With the convention that the initial threshold Th_x for a starting active phase, as no record is processed yet is $Th_x = \frac{1+(1-\delta_x)}{2}$. At the end of each round, every record r_j of an active phase p_x is considered as processed if $\hat{c}_j \leq Th_x$ and the processed records are removed from T_{avl}.

Realization of Workflows. Regardless of the chosen policy, the execution of a workflow always follows the same principles. The structure of workflow W is static and does not change with time. It describes a set of phases $P = \{p_0, \ldots, p_f\}$, their depen-

dencies, and guarded data flows from phases to their successors. A set of n records $R = \{r_1, \ldots, r_n\}$ is used as input to W, i.e., is passed to initial phase p_0, and must be processed with a budget smaller than a given initial budget B_0. As no information about the difficulty of a task d_j^x is available at the beginning of phase p_0, ζ workers are allocated to each record for an initial estimation round. The same principle is followed for each record when it enters a new phase $p_x \in W$. After collection of ζ answers, at each round we first apply EM aggregation to estimate the difficulty d_j^x of active records $r_j \in p_x$, $\hat{c}_j^x$ the confidence in the final aggregated answer y_j^x and the recall α_i and specificity β_i of each worker w_i. Then we use a stopping threshold to decide whether we need more answers for each of the records in a phase. In asynchronous execution, the threshold Th is a global threshold, and in synchronous mode, the confidence of each record r_j in a phase p_x is compared to the local threshold Th_x. Records with sufficient confidence are passed to the next phase(s), for other records we hire new workers to obtain more answers. The decision to start processing records in a phase is immediate in case of asynchronous policy, and delayed until completion of predecessors in the synchronous case. This can increase the confidence level, but also decrease the threshold, as a part of the remaining budget is consumed. The execution stops when the whole budget B_0 is exhausted or when there is no additional record left to process. In the end, the final phase p_f returns the aggregated answer for each of the record.

Termination: One can easily see that as the remaining available budget decreases, the threshold used to decide whether the aggregated answer for a record is final, decreases too. However, there are situations where the confidence in each answer remains low, and the remaining budget reaches 0 before the threshold attains the lower bound 0.5 (that forces moving any record to the next phase(s)) [2]. Similarly, when records do not progress in the workflow, the ratio of remaining work Γ_x remains unchanged for many rounds. As a consequence, the realization of a workflow with our synchronous and asynchronous realization policies may not terminate. We will see in the experimental results section that even with a poor accuracy of workers, this situation was never met. Non-termination corresponds to situations where the weighted answers of workers remain balanced for a long time. The threshold decreases slowly, and the confidence of the aggregated answers remains lower. In that case, when threshold and confidence values eventually coincide (in the worst case at a value of 0.5), the remain-

2. Recall that accepting a confidence threshold of 0.5 amounts to accepting a random guess from workers

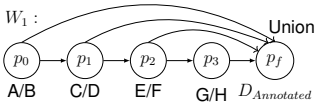
ing budget is too low to realize the remaining work. Solutions to solve this issue and guarantee termination is to bound the sojourn time of a record in a phase, or to keep at every step of execution a sufficient budget to terminate the workflow with a static worker allocation policy hiring only a small number of workers per record. Another way to guarantee termination is to complete the remaining tags with a random guess, which is almost similar to accepting answers with a confidence threshold of 0.5.

6.5 Experiments and results

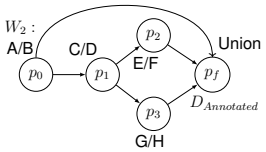
In this section, we evaluate our execution policies on several typical workflows. We consider a standard situation, where a client wants to realize a complex task defined by a workflow on a crowdsourcing platform. The client provides input data and has an initial budget of B_0. As there exists no platform to realize complex tasks, there is no available data to compare the realization of a workflow with our approach to existing complex task executions. To address this issue, we design several typical workflows, synthetic data, and consider realizations of these workflows for various execution policies, characteristics of data, and accuracy of workers.

We consider 5 different workflows, represented in Figure 6.11. Workflow W_1 is a simple sequence of tasks, W_2 is a standard fork-join pattern i.e. parallel processing of data followed by a merge of branches results, W_3 and W_4 are fork-join patterns with equal and different lengths on branches, and W_5 is a more complex workflow with two consecutive forks followed by merges on each branch. Though this set of workflows is not exhaustive, we think it already contains an interesting number of typical patterns met in the execution of complex tasks. We consider that each micro-task simply tags records, and simple exclusive guards sending each record to one successor, depending on the tag obtained at this phase. In Figure 6.11 we depict these choices by pairs of letters (l_0, l_1) representing the binary decision taken on each phase, and assume guards of the form $f == l_0$ or $f == l_1$, where f is the field of records produced by the phase. For example, in workflow W_1, phase p_0 considers two possible tags denoted A and B. After realization of the tasks, if the records are tagged with A by the workers then records are moved to the phase p_f and if tagged with B the records are assigned to phase p_1 for further processing. Each phase of workflows in Figure 6.11 implements similar tagging and decision.

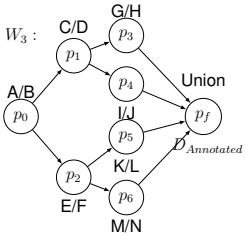
We evaluate average costs and accuracies achieved by workflows realizations with the following parameters. First, the input of each complex task is a dataset of 80 records. Notice that despite this fixed size, the number of micro-tasks to realize by workers vary depending on the execution policy, on the value of data fields produced by workers, but also on the initial dataset, on the initial budget, etc. Each record in the original dataset has initially known data fields, and new fields are added by aggregation of workers answers during the execution of the workflow. For these fields, we assume a prior ground truth, which influences the probability that a worker answers 0 or 1 when filling this field and allows us to simulate an answer by a worker w_i with expertise α_i, β_i. We generate balanced (equal numbers of 0 and 1 in fields) and unbalanced datasets (unbalanced numbers of 0 and 1). Tasks are forwarded according to final answers synthesized from tags returned by the workers. Hence, the balanced and unbalanced characteristics of input dataset influences the number of records that go through a branch or another.



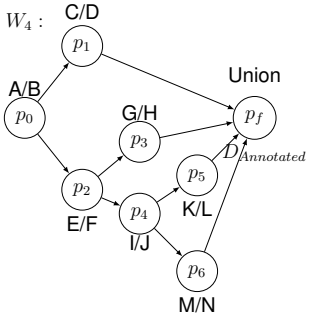
(a) W_1: Sequence of phases.



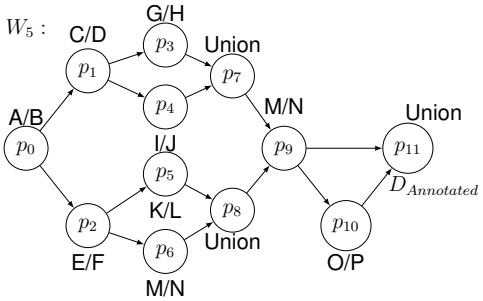
(b) W_2: Parallel data transformations followed by an aggregation of results.



(c) W_3: Fork-join patterns with uniform lengths on their branches.



(d) W_4: Fork-join patterns with nonuniform lengths on their branches.



(e) W_5: Fork-join-Fork patterns with nonuniform lengths on their branches.

Figure 6.11 – Workflows with different orchestrations.

We run the experiment with 4 randomly generated pools of 50 crowd workers, making their accuracy range from low to high expertise. For each pool, we sampled accuracies of workers according to normal distributions ranging respectively in intervals $[0.2, 0.7]$ (low expertise of workers), $[0.4, 0.9]$ (low to average expertise), $[0.6, 0.99]$ (average expertise) and $[0.8, 0.99]$ (high expertise). The composition of pools is shown in Figure 6.12.

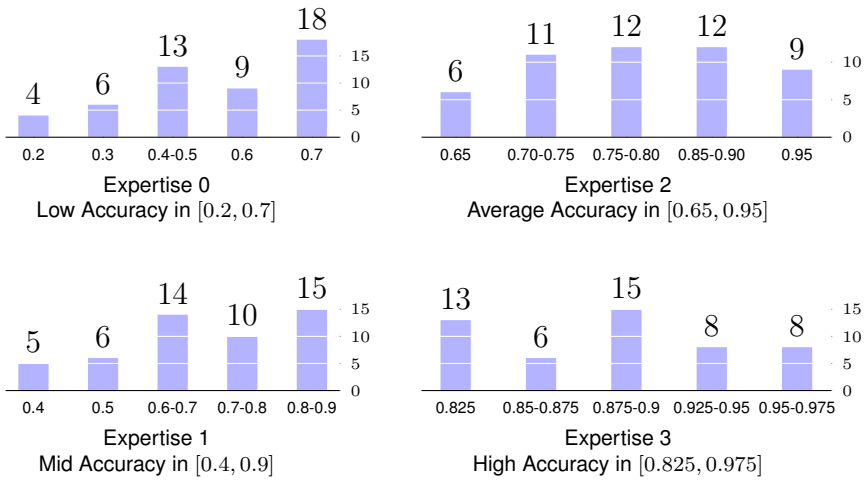


Figure 6.12 – Distributions of workers accuracy(Pool of 50 workers).

The last parameter to set is the initial budget of B_0. We first evaluated the cost for the realization of workflows with a static allocation policy that associates a fixed number of k workers to each record in each phase and aggregates their answers with Majority Voting. We call this policy *Static Majority Voting (static MV)*. For each workflow, we performed random runs with $static\ MV$ to evaluate the maximal budget B_{mv} needed for three different values $k = 10/20/30$. Note that the total budget B_{mv} consumed by $static\ MV$ technique cannot be fixed a priori, as it depends on the execution path followed by records during execution, with random answers of workers. However, static MV is a naive approach showed in the previous chapter, so starting with a budget of $B_0 = B_{mv}$ for realization techniques tailored to save budget is a sensible approach.

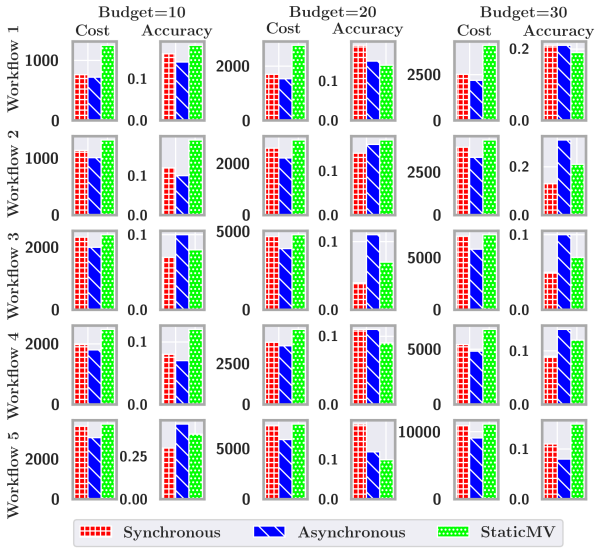
In a second step of the experiment, we used the total budget B_{mv} spent by the static MV approach as an initial budget for synchronous and asynchronous execution. The idea was to achieve at least the same accuracy as static MV with synchronous and asynchronous execution with the same initial budget $B_0 = B_{mv}$, but to spend a smaller fraction of this budget. Overall, our experiments cover the realization of 5 differ-

Workflow	W_1	W_2	W_3	W_4	W_5
Parameter	*Value*				
Workers Accuracy	Low	Mid	Average	High	
Budget for B_{mv}, $k=$	10	20	30		
Data Type	Balanced	Imbalanced			
Mechanisms	Static MV	Synchronous	Asynchronous		

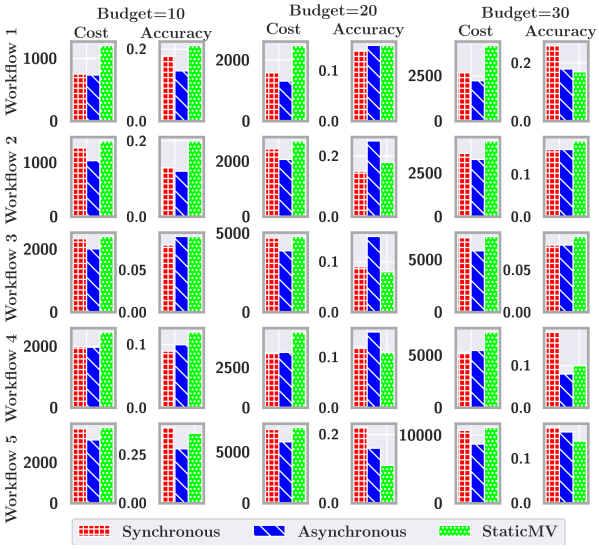
Table 6.1 – Evaluation Parameters.

ent workflows with different values for initial budget, workers accuracy, characteristics of data, and realization policy. This represents 72 different contexts, represented in Table 6.1 (one type of experiment represents a selection of one entry in each row). We run each experiment 15 times to get rid of bias. This represents a sample of 1080 workflow realizations.

We can now analyze the outcomes of our experiments. A first interesting result is that all workflow executions terminated without exhausting their given initial budget, even with low competencies of workers. A second interesting result is that for all realization policies, and for all workflows, complex workflows realization end with poor accuracy when expertise is low i.e. when the distribution of workers expertise is expertise 0 shown in Figure 6.13. Now, consider for instance the results of Figure 6.13. The figure gives the consumed budget and achieved accuracy for a given workflow and a given initial budget when workers have a low expertise. The first series of results concern Workflow 1 with a budget allowing respectively 10, 20, 30 workers per record in each phase when realizing the workflow with static MV policy. The overall expended budget with a static MV approach is around 1200, 2800, 4000, respectively. Regardless of the initial budget, synchronous and asynchronous approaches spend only a fraction of the budget used by static MV. Accuracy is not conclusive, as the best realization policy varies with each experiment: for instance, for W_1 with 10 workers to tag each records in each phase, static MV seems to be the best approach, while with a budget allowing 20 workers per record, the synchronous approach is the best. However, most of the experiments achieve accuracies below 0.2, which is quite low. An explanation is that, as shown in Figure 5.1 in Chapter 5, with low expertise, workers answers are almost random choices. Hence when all workers have low expertise, individual errors are not corrected by other answers, and the ground truth does not influence the results. At each phase, the algorithms take their decisions mostly based on wrong answers provided by the low expert workers, and in consequence, the errors accumulate. The system's behavior is then completely random, which results in poor performance.



(a) Budget spent and accuracy with low expertise of workers and Balanced Data.



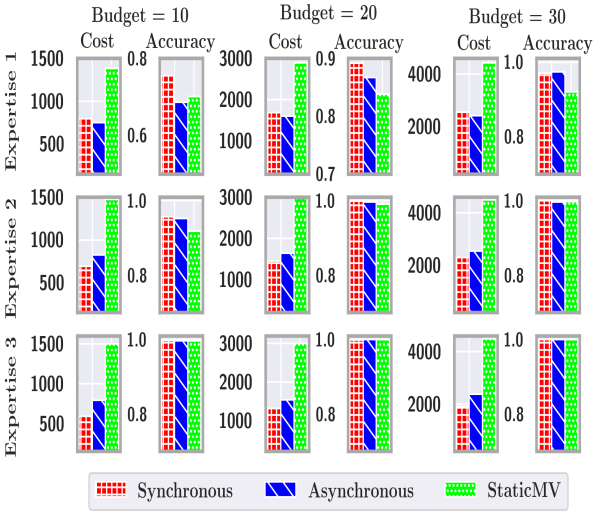
(b) Budget spent and accuracy with low expertise of workers and Unbalanced Data.

Figure 6.13 – Accuracy and cost comparison on low expertise.

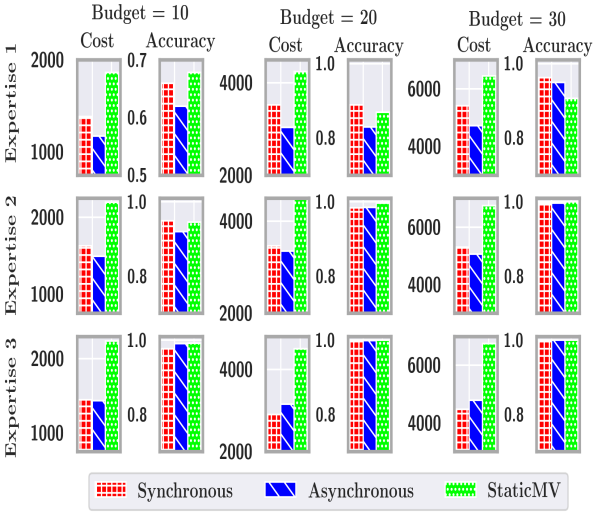
This tendency is shown for all workflows and initial budgets with balanced data (see Figure 6.13a) is confirmed on unbalanced data (see Figure 6.13b).

Next, we consider experiments with mid-level to high expertise, which is the most common case in a crowdsourcing setting. The experiments with competent workers and *synchronous* and *asynchronous* execution policies clearly show that dynamic allocation schemes outperform the static MV approach both in terms of cost and accuracy. One can easily see these results Figures 6.14a, 6.14b, that represent executions of workflow W_1 with three levels of expertise, three initial budgets, and all execution policies, both for balanced and unbalanced data. We get similar results for workflows W_2, W_3, W_4, W_5 as shown in Figures 6.15, 6.16, 6.17, 6.18.

In the worst cases, *synchronous* and *asynchronous* executions achieve accuracies that are almost identical to that of static MV, but in most cases they give answers with better accuracy. With a sufficient initial budget, dynamic approaches achieve accuracy greater than 0.9. An explanation for this improvement of synchronous and asynchronous executions with respect to static MV is that in static MV, one does not consider the expertise of the worker, whereas the *synchronous* and *asynchronous* executions are EM based algorithms that derive the final answers by weighting individual answers

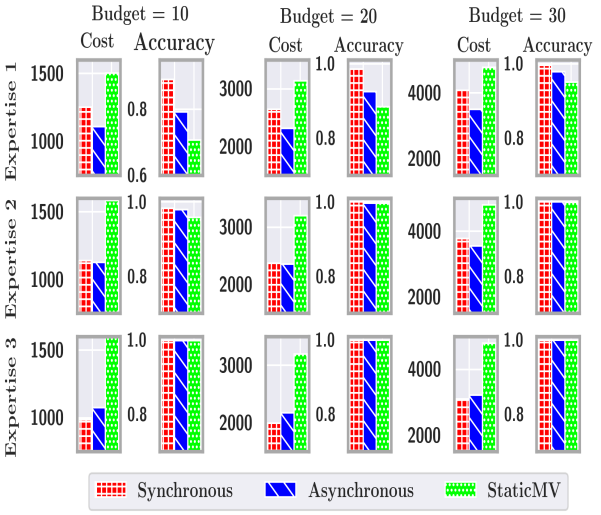


(a) Workflow 1 on Balanced Data.

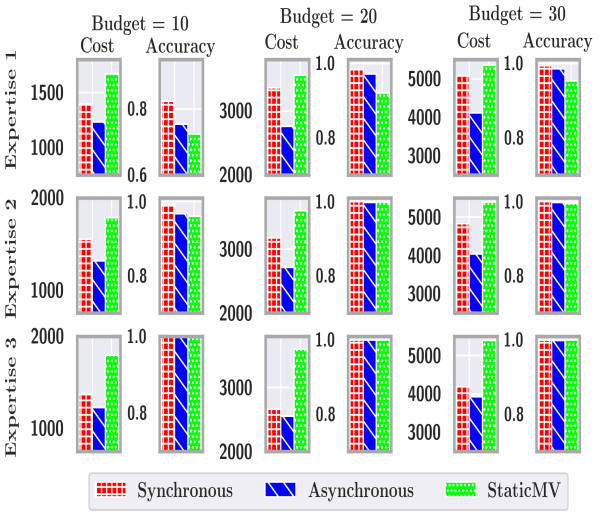


(b) Workflow 1 on Unbalanced Data.

Figure 6.14 – Accuracy and cost comparison on Workflow 1.



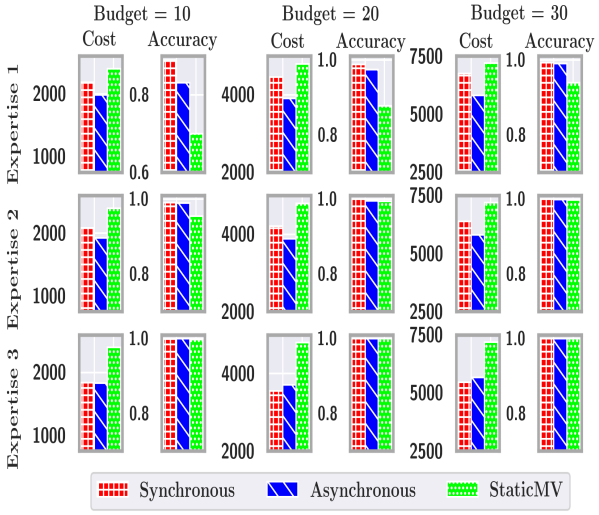
(a) Workflow 2 on Balanced Data.



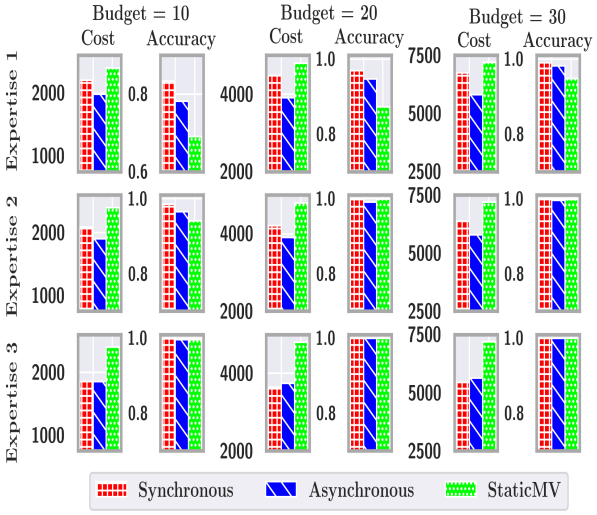
(b) Workflow 2 on Unbalanced Data.

Figure 6.15 – Accuracy and cost comparison on Workflow 2.

according to worker's expertise. This makes EM-based evaluation of final answers more accurate than *static MV*. This improvement already occurs at the level of a single phase execution (this was also the conclusion of Chapter 5).

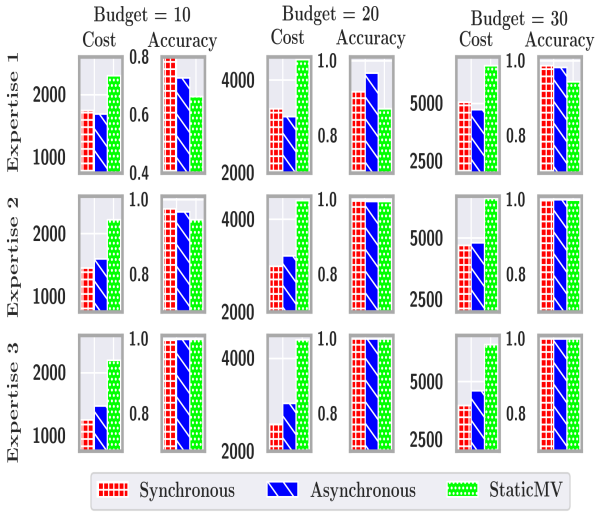


(a) Workflow 3 on Balanced Data.

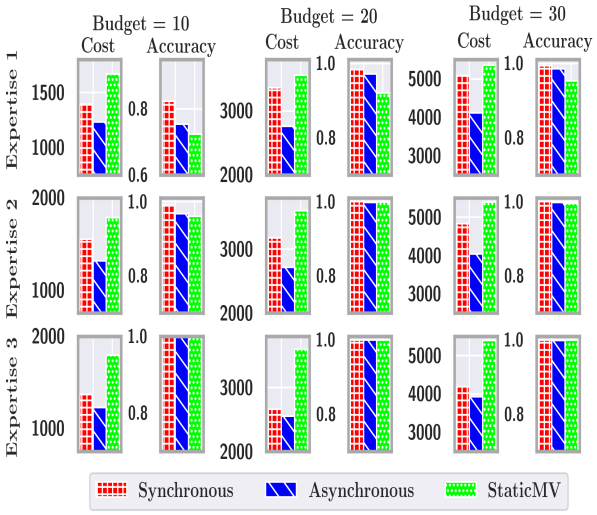


(b) Workflow 3 on Unbalanced Data.

Figure 6.16 – Accuracy and cost comparison on Workflow 3.



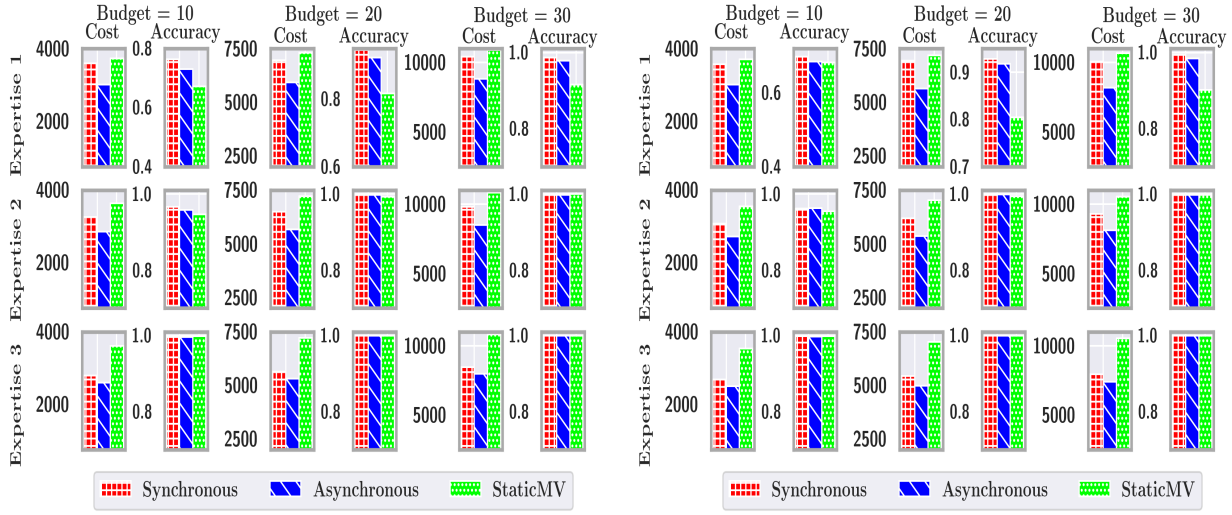
(a) Workflow 4 on Balanced Data.



(b) Workflow 4 on Unbalanced Data.

Figure 6.17 – Accuracy and cost comparison on Workflow 4.

The reasons for cost improvement with respect to *static MV* are also easy to figure. Static MV allocates a fixed number of workers to every record in every phase of a workflow, whereas synchronous and asynchronous execution schemes allocate workers on-the-fly based on a confidence level that depends on the difficulty of tasks, workers expertise, and returned answers. By comparing confidence levels with a dy-



(a) Workflow 5 on Balanced Data.

(b) Workflow 5 on Unbalanced Data.

Figure 6.18 – Accuracy and cost comparison on Workflow 5.

namic threshold, workers allocation considers the remaining budget and workload as well. This clever allocation of workers saves costs, as easy tasks call for the help of fewer workers than the fixed number imposed by static MV. The resources that are not used on easy tasks can be reused later for difficult tasks, hence improving accuracy.

These results were expected. A more surprising outcome of the experiment is that in most cases synchronous execution outperforms the asynchronous execution in terms of accuracy. The intuitive reason behind this result is that the way records are spread in the workflow execution affects the evaluation of expertise and difficulty. The synchronous execution realizes tasks in phases, while asynchronous execution starts tasks independently in the whole workflow. A consequence is that evaluation of hidden variables such as the difficulty of tasks and workers expertise evaluated by the EM aggregation improves with a larger number of records per phase in synchronous execution, while it might remain imprecise when the records are spread in different phases during an asynchronous execution. This precise estimation helps synchronous execution to allocate workers as well as to derive the final answers in a more efficient way and hence outperforms *asynchronous* execution. A third general observation is that both synchronous and asynchronous executions need a greater budget to complete a workflow when data is unbalanced. Observe the results in Figure 6.14a and Figure 6.14b: the budgets spent are always greater with unbalanced data. A possible explanation is that in the balanced cases, records are distributed uniformly on all

phases, which helps evaluation of workers expertise and difficulty of tasks, while with unbalanced data, some phases receive only a few records, which affects the evaluation of hidden variables.

Unsurprisingly (see for instance Figure 6.14a), for a fixed budget, when worker expertise increases, accuracy increases too, and the consumed budget decreases. Competent workers return correct answers, reach a consensus earlier, and hence achieve better accuracy faster. Similarly for a fixed expertise level, increasing the initial budget increases the overall accuracy of the workflow. Again, the explanation is straightforward: a higher budget increases the threshold used to consider an aggregated answer as correct, giving better accuracies. To summarize, for a fixed initial budget and high enough expertise, synchronous and asynchronous policies usually improve both cost and accuracy.

6.6 Conclusion

In this chapter, we proposed a framework to foster the advantages of crowdsourcing systems and of workflow systems. The resulting model can be used to realize complex tasks with the help of a crowd workers. Particular attention is paid to the quality of the data produced, and to the overall cost of complex task realization. We showed the different mechanisms namely synchronous and asynchronous workflow executions and observed how different factors as data, budget, and accuracy affects the overall performance of the workflow. We compared several task distribution strategies through experiments and showed that the dynamic distribution of work outperforms static allocation in terms of cost and accuracy.

A short-term extension for this work is to consider the termination of complex tasks realization with dynamic policies. Indeed, workflows realized with dynamic policies may not terminate: this happens when, for some record, all workers agree to return the answers that do not increase the confidence. However, this situation was never met during our experiments, even with the low expertise of workers. The probability of non-terminating executions with synchronous/asynchronous policies seems negligible. The intuitive reasoning behind this is the threshold falls as the budget is being consumed and it pushes the records to the next phase. An interesting extension is to consider various strategies to hire workers in the most efficient way. Also, in this work, we assumed that all worker are uniformly paid and a non-uniform cost per worker can also

be considered, for instance to hire experts. The difficulty in this case is then to find the best trade-off between costly but accurate answers returned by experts, and cheaper but less accurate answers by general workers. A possibility to address this challenge is to see complex workflows as stochastic games, in which one player tries to maximize accuracy by hiring more expert or costly workers while its opponent tries to hire a large number of workers with lesser costs and the objective is to maximize the accuracy.

PART IV

Closure

CHAPTER 7

CONCLUSION

7.1 Contribution Summary

The objective of this thesis, as stated in the introduction was *to define techniques to deploy complex applications on top of conventional crowdsourcing platforms and to provide data centric algorithms optimizing cost and accuracy.*

The first contribution of the thesis is to define a data centric formal model for complex crowdsourcing applications. The proposed model called *complex workflows* can be used to specify, deploy and verify intricate tasks on top of the existing crowdsourcing platforms. The model provides high-level constructs which allow the design of complex tasks as orchestrations of a set of simple tasks described in the form of a workflow and besides handles worker skills, data dependency, and constraints on data input and output returned by workers. This gives complex workflows a huge expressive power. We also study the termination and correctness properties of the workflow. The second result of the thesis is that existential termination and correctness are undecidable in general. Universal correctness and termination are decidable when constraints and input are specified in a decidable fragment of FO. Last existential correctness and termination become decidable for specifications with bounded recursion which constraints and input are specified in a decidable fragment of FO. However, the complexity of correctness and termination for decidable cases is quite high: they are at least in co-2EXPTIME. This complexity comes from the size of weakest precondition that have to be built to prove termination and correctness.

The third contribution of the thesis is an aggregation technique for crowdsourcing platforms. We considered as key factors the difficulty of tasks, and the expertise of workers, expressed in terms of recall and specificity to model how a worker answers. Aggregation is based on the expectation-maximization algorithm that jointly estimates the

answers, the difficulty of tasks and expertise of workers. Building on this EM based aggregation technique, we propose CrowdInc that optimizes the overall cost to collect answers and aggregate them. The algorithm implements a worker allocation policy that takes decisions from a dynamic threshold computed at each round, which helps in achieving a good trade-off between cost and accuracy. We evaluated the algorithm on existing benchmarks to validate our approach and showed that our aggregation technique outperforms the existing state-of-the-art techniques. We also showed that our incremental crowdsourcing approach achieves the same accuracy as EM with a static allocation of workers, better accuracy than majority voting, and in both cases at lower costs.

The fourth contribution of the thesis is to realize complex workflows with cost and quality assurance. We showed several mechanisms to allocate tasks to workers during the realization of a complex tagging task described by a workflow. We observed that the factors such as data, budget and accuracy of workers affects the overall performance of the workflows realization, and that workers hiring mechanisms also plays an important role in cost minimization and quality maximization. The experiments showed that the dynamic allocation of workers to tasks outperforms the existing static mechanisms in terms of cost and accuracy.

The last contribution of the thesis is the development of the tool CrowdPlex. The tool takes as input a workflow specification, task mapping, profiles, workers availability, workers skills, task pre-requisite, defining an initial configuration and input data. The tool checks termination or correctness of the workflow by deriving weakest preconditions required to satisfy termination or a given FO property along a run. Satisfiability of this precondition is then verified using a SAT solver.

These works have led to the publication of following articles [a, b , c , [d], e].

7.2 Perspectives

In this section, we discuss open research directions. We begin with short term research directions, i.e. the problems that may take less effort and then draw longer term perspectives, i.e. the questions that may take longer time, but possess a potential for high impact.

7.2.1 Short Term Perspectives

Latency. Human latency is one of the critical factors in crowdsourcing. Sometimes workers cannot perform the task that was assigned to them, either because it is too difficult or they are unavailable. Workers come from different and remote places. This means that a workflow can be realized by a crowd composed of workers originating from different time zones. In a complex workflow environment, the tasks are dependent and subsequently, the output of one phase acts as an input to others. In such cases, latency at a particular task delays the workflow. The crowd-powered algorithms need optimization to reduce the latency when eventual results are not available. This calls for definition of lazy mechanisms to progress the workflow based on the partial outcomes: prioritizing for quick partial results by generating some initial results or providing some parameterized approximate output. In a complex crowdsourcing setting, one possible solution to reduce latency is alternative hiring of more workers.

Validation. Workers at crowdsourcing platform are heterogeneous, comes with wide range level of expertise and are unknown. Usually, there is no control on the quality of the workers. Workers at the platform can be spammers which provide answers randomly without looking at the task or provide wrong answers on purpose. Also, some tasks are difficult and are hard to reach consensus. It makes the cost of acquiring answers more expensive and also degrades the overall quality. In *complex workflow* environment where the output of one task acts as an input to other, the wrong answers at a phase directly impact the answerability in the successive phase(s).

The solution to the problem is the post-processing phase in which answers can be validated by experts. In Chapter 5, we give a solution to evaluate the difficulty of each task. The task which are more difficult are those where workers have disagreements and are not able to reach consensus with high confidence. In such cases on top of our model, expert workers can be hired to get their opinion. However, it comes with various other challenges. First, the experts are not so easily available. The hiring of experts also incurs a high cost. The expert varies from domain to domain and in practice it is hard to get experts in each field. Also, experts can be hired at various phases and it requires to identify the best instance in the complex workflow that maximizes the accuracy considering the cost factor. Another solution is to monitor the task execution in the workflows and client or task manager can intervene, guide and suggest the execution of the tasks in a complex workflow. Also, if required the manager can hire the workers

based on the progress of the task. This validation mechanism may need subtle redesign of the system for monitoring the workflow execution and hiring expert workers on demand.

View. In this thesis, we present a data-centric complex workflow model where tasks are individually assigned to the workers. Each worker is provided with data and contribute in its own. The data produced by a worker acts as an input to the other successor tasks. In this setting, workers in complex workflows are only aware of their task, data, operations and executions. They only see a fragment of the whole workflow and have a local view of the system. They are not aware of the global goal of the workflow due to limited access and shortfalls in the understanding of the overall objective of the complex task. Adding a global view to the complex workflow system may bring envision to the overall process. The workers may be able to visualize the final goal and may provide more sensible answers. However, a full view of the workflow may also have issues such as privacy, bias, etc. One of the solutions is to have a global or limited view. The limited visibility can be added in terms of a view of up to some k successor tasks in the workflow. The issue of privacy and bias can be mitigated by data anonymization techniques.

Testing. We tested the *CrowdPlex* tool on various workflow specification with varying workflow, tasks, workers, tasks pre-requisite, worker skills and data specification. The tested use cases range from a very simple to a complex orchestrated workflow. The test of the complex workflow was successful. It allowed to check termination and correctness properties in a few seconds. Next, the tool needs to be tested on real crowdsourcing marketplaces with a more complex environment. The real crowdsourcing comes with a large number of workers as well as varying tasks. The system needs to be scalable in such cases and testing in such an environment will help us to find the limitation of the model. Though, the theoretical worst case complexity demonstrated in Chapter 4 is redhibitory, it is interesting to check if such complexity appears in real situations.

7.2.2 Long Term Perspectives

Collaborative Crowdsourcing. The existing crowdsourcing marketplaces provide a mechanism to realize the tasks independently. Often workers do not collaborate and possess no information about the other sub-tasks in the workflow. Collaborative crowdsourcing is an emerging prototype where a set of workers with diverse and complementary skills can form groups and work together to solve a task. In a collaborative crowdsourcing mechanism, a group of workers works on a single task and contribute individually based on their skills.

To start, Crowd4U [Mor+12] is a volunteer-based system which enables collaborative crowdsourcing with data-flows in a declarative manner. Yet a lot of challenges are still unsolved and collaborative mechanisms comes with diverse challenges. Unlike the independent task execution, collaboration is the principle of such systems. Likewise our complex workflow model, collaborative workflows require an end to end deployment of a task, needs task decomposition, task assignments and effective worker collaboration. Appropriate collaboration schemes are required to enforce efficient coordination between the workers in such systems. Collaborative crowdsourcing can be used for various emerging applications such as article writing, surveillance tasks, text translation, and citizen journalism. Challenges that need to be addressed are worker contribution, sequential or simultaneous collaboration, task-worker assignment, interface design etc.

Imprecision in Data. Workers at crowdsourcing platforms are not always confident about their answers. Sometimes workers make a random guess or provide imprecise results. The existing crowdsourcing systems do not provide ways to enter imprecise results nor provide mechanisms for the self-assessment of answers. The next generation crowdsourcing model should provide flexibility to the workers to enter imprecise data along with the self-assessed confidence score in the provided answers. Such a system can improve answers by getting more imprecise answers for a task which will improve accuracy. Besides, it requires novel aggregation policies. A good starting point could be to consider imprecision and corrective mechanisms based on belief functions. The aggregation policy should consider the imprecise outputs, confidence in answers and then accordingly update the system dynamics to gather more information. Modeling such an imprecise system is not trivial and requires a solid framework to aggregate imprecise answers.

Human in the Loop Machine Learning. Crowdsourcing is a valuable tool to collect annotated data to train machine learning models. However, machine learning models are often not accurate and make mistakes. Human in the loop machine learning is a technique in which a learning algorithm can query a worker to validate the decisions for which it had lesser confidence. The new validated decisions by the crowd workers can then be used to make the machine learning model more precise. The open, dynamic and huge availability of workers round a clock at the crowdsourcing platforms makes a natural choice to leverage the power of the crowd and human wisdom to build more precise machine learning models. For example, consider a machine learning model that classifies an image in categories as cat and dog. Consider, in real test setting for some images the machine learning model is not able to decide with high confidence. In such cases, human intelligence can be leveraged and crowd workers can be hired to annotate the data accurately. The new set of annotated and verified data returned by the workers can then be used as training data to further improve the model decision-making ability. There have been some initial works in this space [Zha+20; Moz+12]. Human in the loop machine learning models can be one of the most mature machine learning paradigm that will continue to attract enough research attention in the highly developing field of artificial intelligence.

Publications

[a] Pierre Bourhis, Loïc Hélouët, Zoltan Miklos, and Rituraj Singh, « Data centric workflows for crowdsourcing », *in*: *International Conference on Applications and Theory of Petri Nets and Concurrency*, Springer, 2020, pp. 24–45 [Regular paper]

[b] Rituraj Singh, Loïc Hélouët, and Zoltan Miklos, « Reducing the Cost of Aggregation in Crowdsourcing », *in*: *International Conference on Web Services*, Springer, 2020, pp. 77–95 [Regular paper]

[c] Pierre Bourhis, Loïc Hélouët, Rituraj Singh, and Zoltán Miklós, « Data Centric Workflows for Crowdsourcing », *in*: (2019) [Full paper]

[d] Rituraj Singh, Loïc Hélouët, and Zoltan Miklos, « Reducing the Cost of Aggregation in Crowdsourcing », *in*: *BDA - Conférence sur la Gestion de Données – Principes, Technologies et Applications*, 2020 [Regular paper]

[e] Loïc Hélouët, Zoltan Miklos, and Rituraj Singh, « Cost and Quality in Crowdsourcing Workflows », *in*: *International Conference on Applications and Theory of Petri Nets and Concurrency*, Springer, 2021, pp. 33–54 [Regular paper]

BIBLIOGRAPHY

[99d] *99Designs*, https://en.99designs.fr/.

[AA11] Mehmet Aci and Mutlu Avci, « K nearest neighbor reinforced expectation maximization method », *in*: *Expert Systems with Applications* 38.*10* (2011), pp. 12585–12591.

[Aal97] Wil MP Van der Aalst, « Verification of workflow nets », *in*: *International Conference on Application and Theory of Petri Nets*, Springer, 1997, pp. 407–426.

[Aal98] Wil MP Van der Aalst, « The application of Petri nets to workflow management », *in*: *Journal of circuits, systems, and computers* 8.*01* (1998), pp. 21–66.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of databases*, vol. 8, Addison-Wesley Reading, 1995.

[All16] Thomas Allweyer, *BPMN 2.0: introduction to the standard for business process modeling*, BoD–Books on Demand, 2016.

[Amta] *Amazon mechanical turk*, https://www.mturk.com/.

[Amtb] *AMT Pricing Policy*, https://www.mturk.com/pricing.

[ASV09] S. Abiteboul, L. Segoufin, and V. Vianu, « Static analysis of active XML systems », *in*: *Trans. on Database Systems* 34.*4* (2009), 23:1–23:44.

[AV13] S. Abiteboul and V. Vianu, « Collaborative data-driven workflows: think global, act local », *in*: *Proc. of PODS'13*, ACM, 2013, pp. 91–102.

[Ayd+14] Bahadir Ismail Aydin, Yavuz Selim Yilmaz, Yaliang Li, Qi Li, Jing Gao, and Murat Demirbas, « Crowdsourcing for multiple-choice question answering », *in*: *Twenty-Sixth IAAI Conference*, 2014.

[Ban+85] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D Ullman, « Magic sets and other strange ways to implement logic programs », *in*: *Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*, 1985, pp. 1–15.

[Bee+06] C. Beeri, A. Eyal, S. Kamenkovich, and Tova Milo, « Querying Business Processes », *in*: *Proc. of VLDB'06*, ACM, 2006, pp. 343–354.

[Beh+11] Tara S Behrend, David J Sharek, Adam W Meade, and Eric N Wiebe, « The viability of crowdsourcing for survey research », *in*: *Behavior research methods* 43.*3* (2011), p. 800.

[Ber+15] M.S. Bernstein, G. Little, R.C. Miller, B. Hartmann, M.S. Ackerman, D.R. Karger, D. Crowell, and K. Panovich, « Soylent: a word processor with a crowd inside », *in*: *Communications of the ACM* 58.*8* (2015), pp. 85–94.

[BHM16] Eric Badouel, Loïc Hélouët, and Christophe Morvan, « Petri nets with structured data », *in*: *Fundamenta Informaticae* 146.*1* (2016), pp. 35–82.

[BHS09] Kamal Bhattacharya, Richard Hull, and Jianwen Su, « A data-centric design methodology for business processes », *in*: *Handbook of Research on Business Process Modeling*, IGI Global, 2009, pp. 503–531.

[Bin] *Microsoft Bing*, `https://www.bing.com/`.

[BK98] Falko Bause and Pieter S Kritzinger, « Stochastic Petri nets: An introduction to the theory », *in*: *ACM SIGMETRICS Performance Evaluation Review* 26.*2* (1998), pp. 2–3.

[Blu02] Moritz Blume, « Expectation maximization: A gentle introduction », *in*: *Technical University of Munich Institute for Computer Science* (2002).

[Boo05] Grady Booch, *The unified modeling language user guide*, Pearson Education India, 2005.

[Bou+19] Pierre Bourhis, Loïc Hélouët, Rituraj Singh, and Zoltán Miklós, « Data Centric Workflows for Crowdsourcing », *in*: (2019).

[Bou+20] Pierre Bourhis, Loïc Hélouët, Zoltan Miklos, and Rituraj Singh, « Data centric workflows for crowdsourcing », *in*: *International Conference on Applications and Theory of Petri Nets and Concurrency*, Springer, 2020, pp. 24–45.

[BS28] P. Bernays and M. Schönfinkel, « Zum entscheidungsproblem der mathematischen logik », *in*: *Mathematische Annalen* 99.*1* (1928), pp. 342–372.

[BT06] D. G. Brizan and A. U. Tansel, « A survey of entity resolution and record linkage methodologies », *in*: *Communications of the IIMA* 6.*3* (2006), p. 5.

[BW18] Jonathan Bragg and Daniel S Weld, « Sprout: Crowd-powered task design for crowdsourcing », *in*: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 2018, pp. 165–176.

[BW84] Alan Bundy and Lincoln Wallen, « Skolemization », *in*: *Catalogue of Artificial Intelligence Tools*, Springer, 1984, pp. 123–123.

[Cao+12] Caleb Chen Cao, Jieying She, Yongxin Tong, and Lei Chen, « Whom to ask ? jury selection for decision making tasks on micro-blog services », *in*: *arXiv preprint arXiv:1208.0273* (2012).

[Car+02] Chad Carson, Serge Belongie, Hayit Greenspan, and Jitendra Malik, « Blobworld: Image segmentation using expectation-maximization and its application to image querying », *in*: *IEEE Transactions on pattern analysis and machine intelligence* 24.*8* (2002), pp. 1026–1038.

[CCAY16] Anand Inasu Chittilappilly, Lei Chen, and Sihem Amer-Yahia, « A survey of general-purpose crowdsourcing techniques », *in*: *IEEE Transactions on Knowledge and Data Engineering* 28.*9* (2016), pp. 2246–2266.

[CDGM13] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali, « Foundations of data-aware process analysis: a database theory perspective », *in*: *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, 2013, pp. 1–12.

[CGT+89] Stefano Ceri, Georg Gottlob, Letizia Tanca, et al., « What you always wanted to know about Datalog(and never dared to ask) », *in*: *IEEE transactions on knowledge and data engineering* 1.*1* (1989), pp. 146–166.

[CGV08] Laura Carnevali, Leonardo Grassi, and Enrico Vicario, « State-density functions over DBM domains in the analysis of non-Markovian models », *in*: *IEEE Transactions on Software Engineering* 35.*2* (2008), pp. 178–194.

[Cha12] Donald D Chamberlin, « Early history of SQL », *in*: *IEEE Annals of the History of Computing* 34.*4* (2012), pp. 78–82.

[Chu+36] Alonzo Church et al., « A note on the Entscheidungsproblem », *in*: *J. Symb. Log.* 1.*1* (1936), pp. 40–41.

[CKO13] Balder ten Cate, Phokion G Kolaitis, and Walied Othman, « Data exchange with arithmetic operations », *in*: *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 537–548.

[Cli] *Clickworker*, `https://www.clickworker.com/`.

[CLZ13] Xi Chen, Qihang Lin, and Dengyong Zhou, « Optimistic knowledge gradient policy for optimal budget allocation in crowdsourcing », *in*: *International conference on machine learning*, 2013, pp. 64–72.

[Cod02] Edgar F Codd, « A relational model of data for large shared data banks », *in*: *Software pioneers*, Springer, 2002, pp. 263–294.

[Cod72] E. F. Codd, « Relational completeness of data base sublanguages », *in*: *Database Systems* (1972), pp. 65–98.

[Cod+72] Edgar F Codd et al., *Relational completeness of data base sublanguages*, Citeseer, 1972.

[Coo+10] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, et al., « Predicting protein structures with a multiplayer online game », *in*: *Nature* 466.*7307* (2010), pp. 756–760.

[Cro] *CrowdFlower*, `http://crowdflower.com/`.

[CSW95] Weidong Chen, Terrance Swift, and David S Warren, « Efficient top-down computation of queries under the well-founded semantics », *in*: *The Journal of logic programming* 24.*3* (1995), pp. 161–199.

[Cze+19] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki, « The reachability problem for Petri nets is not elementary », *in*: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 24–33.

[Dan+18] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh, « Quality Control in Crowdsourcing: A Survey of Quality Attributes, Assessment Techniques, and Assurance Actions », *in*: *ACM Computing Surveys* 51.*1* (2018), p. 7.

[Dav93] Thomas H Davenport, *Process innovation: reengineering work through information technology*, Harvard Business Press, 1993.

[DDCM12] G. Demartini, D.E. Difallah, and Ph. Cudré-Mauroux, « ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking », *in*: *Proc. of WWW 2012*, ACM, 2012, pp. 469–478.

[DDV12] E. Damaggio, A. Deutsch, and V. Vianu, « Artifact systems with data dependencies and arithmetic », *in*: *Trans. on Database Systems* 37.*3* (2012), p. 22.

[Deu+06] A. Deutsch, L. Sui, V. Vianu, and D. Zhou, « Verification of communicating data-driven web services », *in*: *Proc. of PODS'06*, ACM, 2006, pp. 90–99.

[dFM18] M. de Leoni, P. Felli, and M. Montali, « A Holistic Approach for Soundness Verification of Decision-Aware Process Models », *in*: *Proc. of ER'18*, vol. 11157, LNCS, 2018, pp. 219–235.

[DHV14] Alin Deutsch, Richard Hull, and Victor Vianu, « Automatic verification of database-centric systems », *in*: *ACM SIGMOD Record* 43.*3* (2014), pp. 5–17.

[Dij75] E. W. Dijkstra, « Guarded Commands, Nondeterminacy and Formal Derivation of Program. », *in*: *Communications of the ACM* 18.*8* (1975), pp. 453–457.

[DLR77] A.P. Dempster, N. M. Laird, and D. B. Rubin, « Maximum likelihood from incomplete data via the EM algorithm », *in*: *J. of the Royal Statistical Society: Series B (Methodological)* 39.*1* (1977), pp. 1–22.

[DLW13] P. Dai, C. H. Lin, and D. S. Weld, « POMDP-based control of workflows for crowdsourcing », *in*: *Artificial Intelligence* 202 (2013), pp. 52–85.

[DMB08] Leonardo De Moura and Nikolaj Bjørner, « Z3: An efficient SMT solver », *in*: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2008, pp. 337–340.

[DS79] A.Ph. Dawid and A.M. Skene, « Maximum likelihood estimation of observer error-rates using the EM algorithm », *in*: *J. of the Royal Statistical Society: Series C (Applied Statistics)* 28.*1* (1979), pp. 20–28.

[EAGLDG12] Enrique Estellés-Arolas and Fernando González-Ladrón-De-Guevara, « Towards an integrated crowdsourcing definition », *in*: *Journal of Information science* 38.*2* (2012), pp. 189–200.

[Esk+13] Maxine Eskenazi, Gina-Anne Levow, Helen Meng, Gabriel Parent, and David Suendermann, *Crowdsourcing for speech processing: Applications to data collection, transcription and assessment*, John Wiley & Sons, 2013.

[EV11] Carsten Eickhoff and Arjen de Vries, « How crowdsourcable is your task », *in*: *Proceedings of the workshop on crowdsourcing for search and data mining (CSDM) at the fourth ACM international conference on web search and data mining (WSDM)*, 2011, pp. 11–14.

[Fig] *Figure Eight*, `http://www.appen.com`.

[Fin+13] Ailbhe Finnerty, Pavel Kucherbaev, Stefano Tranquillini, and Gregorio Convertino, « Keep it simple: Reward and task design in crowdsourcing », *in*: *Proceedings of the Biannual Conference of the Italian Chapter of SIGCHI*, 2013, pp. 1–4.

[Fla12] P.A. Flach, *Machine Learning - The Art and Science of Algorithms that Make Sense of Data*, Cambridge University Press, 2012.

[FLM19] Paolo Felli, Massimiliano de Leoni, and Marco Montali, « Soundness verification of decision-aware process models with variable-to-variable conditions », *in*: *2019 19th International Conference on Application of Concurrency to System Design (ACSD)*, IEEE, 2019, pp. 82–91.

[Fol] *FoldIt, solve puzzles for science.* `http://fold.it/portal/.`.

[Fow04] Martin Fowler, *UML distilled: a brief guide to the standard object modeling language*, Addison-Wesley Professional, 2004.

[Fra+11] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin, « CrowdDB: answering queries with crowdsourcing », *in*: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 61–72.

[Fre] *Freelancer*, https://www.freelancer.com/.

[Gad+17] Ujwal Gadiraju, Besnik Fetahu, Ricardo Kawase, Patrick Siehndel, and Stefan Dietze, « Using worker self-assessments for competence-based pre-selection in crowdsourcing microtasks », *in*: *ACM Transactions on Computer-Human Interaction (TOCHI)* 24.*4* (2017), pp. 1–26.

[GC11] M.R. Gupta and Y. Chen, « Theory and use of the EM algorithm », *in*: *Foundations and Trends in Signal Processing* 4.*3* (2011), pp. 223–296.

[GIL16] Shinsuke Goto, Toru Ishida, and Donghui Lin, « Understanding Crowdsourcing Workflow: Modeling and Optimizing Iterative and Parallel Processes », *in*: *Proceedings of the Fourth AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2016, 30 October - 3 November, 2016, Austin, Texas, USA*, ed. by Arpita Ghosh and Matthew Lease, AAAI Press, 2016, pp. 52–58.

[Glo] *Global Crowdsourcing Market*, https://www.prnewswire.com/in/news-releases/global-crowdsourcing-market-was-valued-to-be-us-9-519-53-mn-in-2018-and-is-expected-to-reach-us-154-835-74-mn-by-2027-growing-at-a-cagr-of-36-5-over-the-forecast-period-owing-to-advancement-in-crowdsourcing-methods-says-absolute-markets-ins-826488722.html.

[GM05] Robert J Glushko and Tim McGrath, « Document Engineering: analyzing and designing the semantics of Business Service Networks », *in*: *Proceedings of the IEEE EEE05 international workshop on Business services networks*, Citeseer, 2005, pp. 2–2.

[GM+16] H. Garcia-Molina, M. Joglekar, A. Marcus, A. Parameswaran, and V. Verroios, « Challenges in data crowdsourcing », *in*: *Trans. on Knowledge and Data Engineering* 28.*4* (2016), pp. 901–911.

[Goo] *Google Inc.* https://www.google.com/.

[GP14] Yihan Gao and Aditya G. Parameswaran, « Finish Them!: Pricing Algorithms for Human Computation », *in*: *Proc. VLDB Endow.* 7.*14* (2014), pp. 1965–1976.

[Haa+15] Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J. Franklin, « CLAMShell: Speeding up Crowds for Low-latency Data Labeling », *in*: *Proc. VLDB Endow.* 9.*4* (2015), pp. 372–383.

[Hac76] Michel Henri Théodore Hack, « Decidability questions for Petri Nets. », PhD thesis, Massachusetts Institute of Technology, 1976.

[Har+13] B.B Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali, « Verification of relational data-centric dynamic systems with external services », *in*: *Proc. of PODS 2013*, 2013, pp. 163–174.

[HGL11] Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo, « Datalog and emerging applications: an interactive tutorial », *in*: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 1213–1216.

[Hil+95] Gerd G Hillebrand, Paris C Kanellakis, Harry G Mairson, and Moshe Y Vardi, « Undecidable boundedness problems for datalog programs », *in*: *The Journal of logic programming* 25.*2* (1995), pp. 163–190.

[HMS21] Loïc Hélouët, Zoltan Miklos, and Rituraj Singh, « Cost and Quality in Crowdsourcing Workflows », *in*: *International Conference on Applications and Theory of Petri Nets and Concurrency*, Springer, 2021, pp. 33–54.

[Hoa69] Charles Antony Richard Hoare, « An axiomatic basis for computer programming », *in*: *Communications of the ACM* 12.*10* (1969), pp. 576–580.

[Hor+12] András Horváth, Marco Paolieri, Lorenzo Ridi, and Enrico Vicario, « Transient analysis of non-Markovian models using stochastic state classes », *in*: *Performance Evaluation* 69.*7-8* (2012), pp. 315–335.

[How06] Jeff Howe, « The rise of crowdsourcing », *in*: *Wired magazine* 14.*6* (2006), pp. 1–4.

[Hul+99] Richard Hull, Francois Llirbat, Eric Siman, Jianwen Su, Guozhu Dong, Bharat Kumar, and Gang Zhou, « Declarative workflows that support easy modification and dynamic browsing », *in*: *ACM SIGSOFT Software Engineering Notes* 24.*2* (1999), pp. 69–78.

[HZS17] Simone Hantke, Zixing Zhang, and Björn W Schuller, « Towards Intelligent Crowdsourcing for Audio Data Annotation: Integrating Active Learning in the Real World. », *in*: *INTERSPEECH*, 2017, pp. 3951–3955.

[Ipe10] Panagiotis G Ipeirotis, « Analyzing the amazon mechanical turk marketplace », *in*: *XRDS: Crossroads, The ACM Magazine for Students* 17.*2* (2010), pp. 16–21.

[Itz+17] S. Itzhaky, T. Kotek, N. Rinetzky, M. Sagiv, O. Tamir, H. Veith, and F. Zuleger, « On the Automated Verification of Web Applications with Embedded SQL », *in*: *Proc. of ICDT'17*, vol. 68, LIPIcs, 2017, 16:1–16:18.

[Jen89] K. Jensen, « Coloured Petri nets: A high level language for system design and analysis », *in*: *Proc. of Petri Nets'90*, vol. 483, LNCS, 1989, pp. 342–416.

[Jor+07] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, et al., « Web services business process execution language version 2.0 », *in*: *OASIS standard* 11.*120* (2007), p. 5.

[Kam68] Johan Anthony Wilem Kamp, « Tense logic and the theory of linear order », *in*: (1968).

[KCH12] A. Kulkarni, M. Can, and B. Hartmann, « Collaboratively crowdsourcing workflows with turkomatic », *in*: *Proc. of CSCW'12*, ACM, 2012, pp. 1003–1012.

[KCM06] D. Kitchin, W.R. Cook, and J. Misra, « A Language for Task Orchestration and Its Semantic Properties », *in*: *Proc. of CONCUR'06*, 2006, pp. 477–491.

[KG12] Hyun-Chul Kim and Zoubin Ghahramani, « Bayesian classifier combination », *in*: *Artificial Intelligence and Statistics*, 2012, pp. 619–627.

[Kha11] Mumit Khan, *Longest path in a directed acyclic graph (DAG)*, 2011.

[Kha+11] Firas Khatib, Frank DiMaio, Seth Cooper, Maciej Kazmierczyk, Miroslaw Gilski, Szymon Krzywda, Helena Zabranska, Iva Pichova, James Thompson, Zoran Popović, et al., « Crystal structure of a monomeric retroviral protease solved by protein folding game players », *in*: *Nature structural & molecular biology* 18.*10* (2011), pp. 1175–1177.

[Kha80] L.G. Khashiyan, « Polynomial algorithms in linear programming », *in*: *U.S.S.R. Computational Mathematics and Mathematical Physics* 20 (1980), pp. 51–68.

[Kit+11] A. Kittur, B. Smus, S. Khamkar, and R.E. Kraut, « Crowdforge: Crowdsourcing complex work », *in*: *Proc. of UIST'11*, ACM, 2011, pp. 43–52.

[Kit+12] Aniket Kittur, Susheel Khamkar, Paul André, and Robert Kraut, « CrowdWeaver: visually managing complex crowd work », *in*: *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, 2012, pp. 1033–1036.

[KM69] Richard M Karp and Raymond E Miller, « Parallel program schemata », *in*: *Journal of Computer and system Sciences* 3.*2* (1969), pp. 147–195.

[KOS11] D.R. Karger, S. Oh, and D. Shah, « Iterative learning for reliable crowdsourcing systems », *in*: *Proc. of NIPS'11*, 2011, pp. 1953–1961.

[Kry18] Kryll, *A powerful crypto trading tool for everyone*, tech. rep., `https://twitter.com/kryll_io`, KRYLL, 2018.

[KSK16] Aikaterini Katmada, Anna Satsiou, and Ioannis Kompatsiaris, « Incentive mechanisms for crowdsourcing platforms », *in*: *International Conference on Internet Science*, Springer, 2016, pp. 3–18.

[Kuc+16] P. Kucherbaev, F. Daniel, S. Tranquillini, and M. Marchese, « Crowdsourcing processes: A survey of approaches and opportunities », *in*: *IEEE Internet Computing* 20.*2* (2016), pp. 50–56.

[Kum+03] Santhosh Kumaran, Prabir Nandi, Terry Heath, Kumar Bhaskaran, and Raja Das, « ADoc-oriented programming », *in*: *2003 Symposium on Applications and the Internet, 2003. Proceedings.* IEEE, 2003, pp. 334–341.

[KV17] A. Koutsos and V. Vianu, « Process-centric views of data-driven business artifacts », *in*: *Journal of Computer and System Sciences* 86 (2017), pp. 82–107.

[KWD10] Ehud D Karnin, Eugene Walach, and Tal Drory, « Crowdsourcing in the document processing practice », *in*: *International Conference on Web Engineering*, Springer, 2010, pp. 408–411.

[Laz+08] Ranko Lazić, Tom Newcomb, Joël Ouaknine, Andrew W Roscoe, and James Worrell, « Nets with tokens which carry data », *in*: *Fundamenta Informaticae* 88.*3* (2008), pp. 251–274.

[LB88] Mary J Lindstrom and Douglas M Bates, « Newton—Raphson and EM algorithms for linear mixed-effects models for repeated-measures data », *in*: *Journal of the American Statistical Association* 83.*404* (1988), pp. 1014–1022.

[Le+10] J. Le, A. Edmonds, V. Hester, and L. Biewald, « Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution », *in*: *SIGIR 2010 workshop on crowdsourcing for search evaluation*, vol. 2126, 2010, pp. 22–32.

[Lea11] Matthew Lease, « On quality control and machine learning in crowdsourcing », *in*: *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[Leo+14] Massimiliano de Leoni, Jorge Munoz-Gama, Josep Carmona, and Wil MP van der Aalst, « Decomposing alignment-based conformance checking of data-aware process models », *in*: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2014, pp. 3–20.

[Lew80] H.R. Lewis, « Complexity Results for Classes of Quantificational Formulas », *in*: *J. Comput. Syst. Sci.* 21.*3* (1980), pp. 317–353.

[LGC15] Ajeet Lakhani, Ashish Gupta, and K Chandrasekaran, « IntelliSearch: A search engine based on Big Data analytics integrated with crowdsourcing and category-based search », *in*: *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, IEEE, 2015, pp. 1–6.

[Li+14] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han, « Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation », *in*: *Proc. of SIGMOD'14*, ACM, 2014, pp. 1187–1198.

[Li+16a] G. Li, J. Wang, Y. Zheng, and M.J. Franklin, « Crowdsourced data management: A survey », *in*: *Trans. on Knowledge and Data Engineering* 28.*9* (2016), pp. 2296–2319.

[Li+16b] Qi Li, Fenglong Ma, Jing Gao, Lu Su, and Christopher J Quinn, « Crowdsourcing high quality labels with a tight budget », *in*: *Proceedings of the ninth acm international conference on web search and data mining*, 2016, pp. 237–246.

[Lit+09] G. Little, L.B. Chilton, M. Goldman, and R.C. Miller, « Turkit: tools for iterative tasks on Mechanical Turk », *in*: *Proc. of HCOMP'09*, ACM, 2009, pp. 29–30.

[Liu12] B. Liu, « Sentiment analysis and opinion mining », *in*: *Synthesis lectures on human language technologies* 5.*1* (2012), pp. 1–167.

[Lom01] Irina A Lomazova, « Nested Petri nets: Multi-level and recursive systems », *in*: *Fundamenta Informaticae* 47.*3-4* (2001), pp. 283–293.

[Löw15] L. Löwenheim, « Über möglichkeiten im relativkalkül », *in*:*Math. Ann.* 76.*4* (1915), pp. 447–470.

[LS00] Sea Ling and Heinz Schmidt, « Time Petri nets for workflow modelling and analysis », *in*: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, vol. 4, IEEE, 2000, pp. 3039–3044.

[LS99] I.A. Lomazova and Ph. Schnoebelen, « Some Decidability Results for Nested Petri Nets », *in*: *Perspectives of System Informatics*, 1999, pp. 208–220.

[Mao+17] Ke Mao, Licia Capra, Mark Harman, and Yue Jia, « A survey of the use of crowdsourcing in software engineering », *in*: *Journal of Systems and Software* 126 (2017), pp. 57–84.

[May81] Ernst Mayr, « Persistence of vector replacement systems is decidable », *in*: *Acta Informatica* 15.*3* (1981), pp. 309–318.

[Mei+20] Hongyuan Mei, Guanghui Qin, Minjie Xu, and Jason Eisner, « Neural Datalog through time: Informed temporal modeling via logical specification », *in*: *International Conference on Machine Learning*, PMLR, 2020, pp. 6808–6819.

[Mer74] Philip Merlin, « A study of the recoverability of communication protocols », *in*: *PhD Theses, Irvine* (1974).

[MGAM16] Panagiotis Mavridis, David Gross-Amblard, and Zoltán Miklós, « Using hierarchical skills for optimized task assignment in knowledge-intensive crowdsourcing », *in*: *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 843–853.

[MGM16] P. Mavridis, D. Gross-Amblard, and Z. Miklós, « Using Hierarchical Skills for Optimized Task Assignment in Knowledge-Intensive Crowdsourcing », *in*: *Proc. of WWW'16*, ACM, 2016, pp. 843–853.

[Min67] Marvin Lee Minsky, *Computation*, Prentice-Hall Englewood Cliffs, 1967.

[Mol82] Michael K. Molloy, « Performance analysis using stochastic Petri nets », *in*: *IEEE Transactions on computers 9* (1982), pp. 913–917.

[Moo96] Todd K Moon, « The expectation-maximization algorithm », *in*: *IEEE Signal processing magazine* 13.*6* (1996), pp. 47–60.

[Mor+12] Atsuyuki Morishima, Norihide Shinagawa, Tomomi Mitsuishi, Hideto Aoki, and Shun Fukusumi, « CyLog/Crowd4U: A declarative platform for complex data-centric crowdsourcing », *in*: *Proceedings of the VLDB Endowment* 5.*12* (2012), pp. 1918–1921.

[Mor19] Mohammad Moradi, « Crowdsourcing for search engines: perspectives and challenges », *in*: *International Journal of Crowd Science* (2019).

[Mor75] M. Mortimer, « On languages with two variables », *in*: *Mathematical Logic Quarterly* 21.*1* (1975), pp. 135–140.

[Moz+12] Barzan Mozafari, Purnamrita Sarkar, Michael J Franklin, Michael I Jordan, and Samuel Madden, « Active learning for crowd-sourced databases », *in*: *arXiv preprint arXiv:1209.3686* (2012).

[Mur89] Tadao Murata, « Petri nets: Properties, analysis and applications », *in*: *Proceedings of the IEEE* 77.*4* (1989), pp. 541–580.

[Nac] *PetriNets Tutorial*, http://www.cs.tau.ac.il/~nachumd/models/Nets.pdf.

[Nas84] S. G. Nash, « Newton-type minimization via the Lanczos method », *in*: *SIAM J. on Numerical Analysis* 21.*4* (1984), pp. 770–788.

[NC03] A. Nigam and N.S. Caswell, « Business artifacts: An approach to operational specification », *in*: *IBM Systems Journal* 42.*3* (2003), pp. 428–445.

[NW06] J. Nocedal and S. Wright, « Numerical optimization: Springer science & business media », *in*: *New York* (2006).

[OAS07] OASIS, *Web Services Business Process Execution Language*, tech. rep., http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, OASIS, 2007.

[Par+12] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom, « Deco: declarative crowdsourcing », *in*: *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 1203–1212.

[PCZ15] Shenle Pan, Chao Chen, and Ray Y Zhong, « A crowdsourcing solution to collect e-commerce reverse flows in metropolitan areas », *in*: *IFAC-PapersOnLine* 48.*3* (2015), pp. 1984–1989.

[Pet62] Carl Adam Petri, « Kommunikation mit automaten », *in*: (1962).

[PL04] B. Pang and L. Lee, « A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts », *in*: *Proc. of the 42nd annual meeting on Association for Computational Linguistics*, Association for Computational Linguistics, 2004, p. 271.

[Pnu77] Amir Pnueli, « The temporal logic of programs », *in*: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, IEEE, 1977, pp. 46–57.

[Pro] *Programming Z3*, http://theory.stanford.edu/~nikolaj/programmingz3.html.

[QB11] Alexander J Quinn and Benjamin B Bederson, « Human computation: a survey and taxonomy of a growing field », *in*: *Proceedings of the SIGCHI conference on human factors in computing systems*, 2011, pp. 1403–1412.

[Quo] *Quora*, `https://www.quora.com/`.

[RA14] V. Raykar and P. Agrawal, « Sequential crowdsourced labeling as an epsilon-greedy exploration in a Markov Decision Process », *in*: *Artificial intelligence and statistics*, 2014, pp. 832–840.

[Ray+10] V. C. Raykar, S. Yu, L.H. Zhao, G.H. Valadez, C. Florin, L. Bogoni, and L. Moy, « Learning from crowds », *in*: *J. of Machine Learning Research* 11.*Apr* (2010), pp. 1297–1322.

[Rec+03] Laura Recalde, Manuel Silva, Joaquín Ezpeleta, and Enrique Teruel, « Petri nets and manufacturing systems: An examples-driven tour », *in*: *Advanced Course on Petri Nets*, Springer, 2003, pp. 742–788.

[Ric68] Daniel Richardson, « Some undecidable problems involving elementary functions of a real variable », *in*: *The Journal of Symbolic Logic* 33.*4* (1968), pp. 514–520.

[Ros+10] Joel Ross, Lilly Irani, M Six Silberman, Andrew Zaldivar, and Bill Tomlinson, « Who are the crowdworkers? Shifting demographics in Mechanical Turk », *in*: *CHI'10 extended abstracts on Human factors in computing systems*, 2010, pp. 2863–2872.

[SC+15] D. Sánchez-Charles, V. Muntés-Mulero, M. Solé, and J. Nin, « CrowdWON: A Modelling Language for Crowd Processes based on Workflow Nets. », *in*: *AAAI*, 2015, pp. 1284–1290.

[Sch13] Florian Alexander Schmidt, « The good, the bad and the ugly: Why crowdsourcing needs ethics », *in*: *2013 International Conference on Cloud and Green Computing*, IEEE, 2013, pp. 531–535.

[SDS20] Asit Subudhi, Manasa Dash, and Sukanta Sabut, « Automated segmentation and classification of brain stroke using expectation-maximization and random forest classifier », *in*: *Biocybernetics and Biomedical Engineering* 40.*1* (2020), pp. 277–289.

[SHM20a] Rituraj Singh, Loïc Hélouët, and Zoltan Miklos, « Reducing the Cost of Aggregation in Crowdsourcing », *in*: *International Conference on Web Services*, Springer, 2020, pp. 77–95.

[SHM20b] Rituraj Singh, Loïc Hélouët, and Zoltan Miklos, « Reducing the Cost of Aggregation in Crowdsourcing », *in*: *BDA - Conférence sur la Gestion de Données – Principes, Technologies et Applications*, 2020.

[Sta] *StackExchange*, `https://stackexchange.com/`.

[SVW16] T. Sturm, M. Voigt, and C. Weidenbach, « Deciding First-Order Satisfiability when Universal and Existential Variables are Separated », *in*: *Proc. of LICS'16*, 2016, pp. 86–95.

[TAK20] Zeinab Tirandaz, Gholamreza Akbarizadeh, and Hooman Kaabi, « PolSAR image segmentation based on feature extraction and data compression using Weighted Neighborhood Filter Bank and Hidden Markov random field-expectation maximization », *in*: *Measurement* 153 (2020), p. 107432.

[Tar98] Alfred Tarski, « A decision method for elementary algebra and geometry », *in*: *Quantifier elimination and cylindrical algebraic decomposition*, Springer, 1998, pp. 24–84.

[ŢM05] Ferucio Laurenţiu Ţiplea and Dan Cristian Marinescu, « Structural soundness of workflow nets is decidable », *in*: *Information Processing Letters* 96.*2* (2005), pp. 54–58.

[Tra+15] S. Tranquillini, F. Daniel, P. Kucherbaev, and F. Casati, « Modeling, Enacting, and Integrating Custom Crowdsourcing Processes », *in*: *TWEB* 9.*2* (2015), 7:1–7:43.

[TT+13] L. Tran-Thanh, M. Venanzi, A. Rogers, and N.R. Jennings, « Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks », *in*: *Proc. of AAMAS'13*, International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 901–908.

[TT+14] Long Tran-Thanh, Trung Dong Huynh, Avi Rosenfeld, Sarvapali Ramchurn, and Nicholas R Jennings, « BudgetFix: budget limited crowdsourcing for interdependent task allocation with quality guarantees », *in*: (2014).

[TT+15] Long Tran-Thanh, Trung Dong Huynh, Avi Rosenfeld, Sarvapali D Ramchurn, and Nicholas R Jennings, « Crowdsourcing complex workflows under budget constraints », *in*: *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Tur37] Alan Mathison Turing, *On computable numbers, with an application to the Entscheidungsproblem; a correction*, Royal Society, 1937.

[Upw] *Upwork, In-demand talent on demand.* `https://www.upwork.com/`.

[VD00] David A Van Dyk, « Fitting mixed-effects models using efficient EM-type algorithms », *in*: *Journal of Computational and Graphical Statistics* 9.*1* (2000), pp. 78–98.

[VDA+11] W.M.P Van Der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, HMW Verbeek, M. Voorhoeve, and M.T. Wynn, « Soundness of workflow nets: classification, decidability, and analysis », *in*: *Formal Aspects of Computing* 23.*3* (2011), pp. 333–363.

[VDAVHH04] Wil Van Der Aalst, Kees Max Van Hee, and Kees van Hee, *Workflow management: models, methods, and systems*, MIT press, 2004.

[Ven+14] M. Venanzi, J. Guiver, G. Kazai, P. Kohli, and M. Shokouhi, « Community-based bayesian aggregation models for crowdsourcing », *in*: *Proc. of WWW'14*, ACM, 2014, pp. 155–164.

[VHSV04] Kees Van Hee, Natalia Sidorova, and Marc Voorhoeve, « Generalised soundness of workflow nets is decidable », *in*: *International Conference on Application and Theory of Petri Nets*, Springer, 2004, pp. 197–215.

[Wan+12] J. Wang, T. Kraska, M.J. Franklin, and J. Feng, « Crowder: Crowdsourcing entity resolution », *in*: *Proc. of the VLDB Endowment* 5.*11* (2012), pp. 1483–1494.

[Wel+10] P. Welinder, S. Branson, P. Perona, and S.J. Belongie, « The multidimensional wisdom of crowds », *in*: *Proc. of NIPS'10*, 2010, pp. 2424–2432.

[Wes12] Mathias Weske, « Business process management architectures », *in*: *Business Process Management*, Springer, 2012, pp. 333–371.

[Whi04] Stephen A White, « Process modeling notations and workflow patterns », *in*: *Workflow handbook* 2004 (2004), pp. 265–294.

[Whi+09] J. Whitehill, T. Wu, J. Bergsma, J.R. Movellan, and P.L. Ruvolo, « Whose vote should count more: Optimal integration of labels from labelers of unknown expertise », *in*: *Proc. of NIPS'09*, 2009, pp. 2035–2043.

[Wir] *Wirk*, `https://www.wirk.io/`.

[WK05] Jianrui Wang and Akhil Kumar, « A framework for document-driven workflow systems », *in*: *International Conference on Business Process Management*, Springer, 2005, pp. 285–301.

[WRA20] Dong Wei, Senjuti Basu Roy, and Sihem Amer-Yahia, « Recommending Deployment Strategies for Collaborative Tasks », *in*: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, ed. by David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, ACM, 2020, pp. 3–17.

[Wu+15] Heting Wu, Hailong Sun, Yili Fang, Kefan Hu, Yongqing Xie, Yangqiu Song, and Xudong Liu, « Combining machine learning and crowdsourcing for better understanding commodity reviews », *in*: *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[XFT15] Aifang Xu, Xiaonan Feng, and Ye Tian, « Revealing, characterizing, and detecting crowdsourcing spammers: A case study in community Q&A », *in*: *2015 IEEE Conference on Computer Communications (INFOCOM)*, IEEE, 2015, pp. 2533–2541.

[Yah] *Yahoo! Inc*, `http://www.yahoo.com.`.

[Yu+12] Han Yu, Zhiqi Shen, Chunyan Miao, and Bo An, « Challenges and opportunities for trust management in crowdsourcing », *in*: *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, vol. 2, IEEE, 2012, pp. 486–493.

[YWL15] Bin Ye, Yan Wang, and Ling Liu, « Crowd trust: A context-aware trust model for worker selection in crowdsourcing environments », *in*: *2015 IEEE international conference on web services*, IEEE, 2015, pp. 121–128.

[Zha+20] Yunpeng Zhao, Mattia Prosperi, Tianchen Lyu, Yi Guo, and Jing Bian, « Integrating Crowdsourcing and Active Learning for Classification of Work-Life Events from Tweets », *in*: *arXiv preprint arXiv:2003.12139* (2020).

[Zhe+16] Q. Zheng, W. Wang, Y. Yu, M. Pan, and X. Shi, « Crowdsourcing Complex Task Automatically by Workflow Technology », *in*: *MiPAC'16 Workshop*, 2016, pp. 17–30.

[Zhe+17] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng, « Truth inference in crowdsourcing: Is the problem solved? », *in*: *Proc. of the VLDB Endowment* 10.*5* (2017), pp. 541–552.

[ZLH11] Haichao Zheng, Dahui Li, and Wenhua Hou, « Task design, motivation, and participation in crowdsourcing contests », *in*: *International Journal of Electronic Commerce* 15.*4* (2011), pp. 57–88.

[ZLM14] Dong Zhao, Xiang-Yang Li, and Huadong Ma, « How to crowdsource tasks truthfully without sacrificing utility: Online incentive mechanisms with budget constraint », *in*: *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, IEEE, 2014, pp. 1213–1221.

APPENDIX

A.1 Proof of Theorem 1

Theorem 1 Existential termination of complex workflows is undecidable.

Proof. The proof is done by reduction from the halting problem of two counter machines to termination of complex workflows. A 2-counter machine (2CM) is a tuple $\langle Q, c_1, c_2, I, q_0, q_f \rangle$ where:

— Q is a finite set of states.

— $q_0 \in Q$ is the initial state, $q_f \in Q$ is a particular state called the final state.

— c_1, c_2 are two counters holding non-negative integers.

— $I = I_1 \cup I_2$ is a set of instructions. Instructions in I_1 are of the form $inst_q = inc(q, c_l, q')$, depicting the fact that the machine is in state q, increases the value of counter c_l by 1, and moves to a new state q'. Instructions in I_2 are of the form $inst_q = dec(q, c_l, q', q'')$, depicting the fact that the machine is in state q, if $c_l == 0$, the machine moves to new state q' without making any change in the value of counter c_l, and otherwise, decrements the counter c_l and moves to state q''. We consider deterministic machines, i.e. there is at most one instruction $inst_q$ per state in $I_1 \cup I_2$. At any instant, the machine is in a configuration $C = (q, v_1, v_2)$ where q is the current state, v_1 the value of counter c_1 and v_2 the value of counter c_2. The machine executes instructions from its current configuration, and stops as soon as it reaches state q_f.

From a given configuration $C = (q, v_1, v_2)$, a machine can only execute instruction $inst_q$, and hence the next configuration denoted by $\Delta(C)$ of the machine is also unique. A run of a two counters machine is a sequence of configurations $\rho = C_0.C_1 \ldots C_k$ such that $C_i = \Delta(C_{i-1})$. The *halting problem* is defined as follows: given a 2-CM, an initial configuration $C_0 = (q_0, 0, 0)$, decide whether a run of the machine reaches some configuration (q_f, n_1, n_2), where q_f is the final state and n_1, n_2 are arbitrary values of the counter. It is well known that this halting problem is undecidable [Min67].

Let us now show how to encode a counter machine with complex workflows.

— We consider a dataset D with relational schema $rs = (R, \{k, cname\})$ where k

is a unique identifier, and $cname \in Cnt_1, Cnt_2, \perp$. Clearly, we can encode the value of counter c_x with the cardinal of $\{(k, n) \in D \mid n = Cnt_x\}$. We start from a configuration where the dataset contains a single record $R(0, \perp)$

— For every instruction of the form $inc(q, c_x, q')$, we create a task t_q, and a workflow W_q^{inc}, and a worker u_q, who is the only worker allowed to execute this task. The only operation that u_q can do is refine t_q with workflow W_q^{inc}. W_q^{inc} has two nodes n_q^{inc} and $n_{q'}$ such that $(n_q^{inc}, n_{q'}) \in \longrightarrow$, $\lambda(n_q^{inc}) = t_q^{inc}$ and $\lambda(n_{q'}) = t_{q'}$ as depicted in Figure A.1. Task t_q^{inc} is an atomic task that adds one record of the form (k', Cnt_x) to the dataset. Hence, after executing tasks t_q and t_q^{inc}, the number of occurrences of Cnt_x has increased by one.

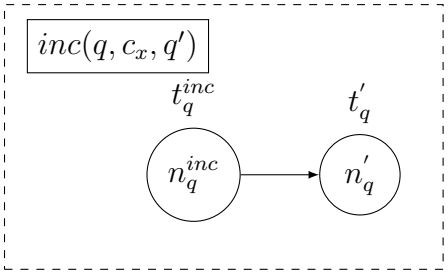


Figure A.1 – Encoding of $inc(q, c_x, q')$ instruction.

— For every instruction of the form $dec(q, c_x, q', q'')$, we create a complex task t_q and a worker u_q who can choose to refine t_q according to rule $(t_q, W_{q,Z})$ or rule $(t_q, , W_{q,NZ})$ as shown in Figure A.2. The choice of one workflow or another will simulate the decision to perform a zero test or a non-zero test. Note that as the choice of a particular rule to replace a task workflow is non-deterministic, worker u_q can choose one or the other. However, the wrong choice of refinement will cause a deadlock.

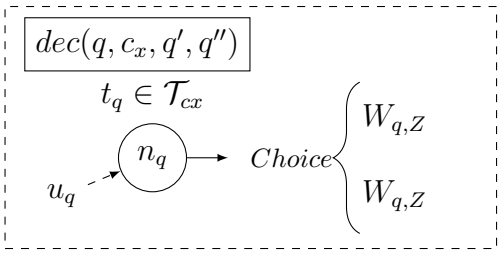


Figure A.2 – Encoding of $dec(q, c_x, q', q'')$ instruction.

— Let us now detail the contents of $W_{q,NZ}$, represented in Figure A.3. This workflow is composed of nodes $n_q^{div}, n_q^{C_x}, n_q^{C_{\bar{x}} \cup \perp}, n_q^{\otimes}, n_q^{dec}$ and $n_{q''}$, respectively labeled by

tasks $t_q^{div}, t_q^{C_x}, t_q^{C_{\bar{x}}\cup\perp}, t_q^{\otimes}$, t_q^{dec} and $t_{q''}$. The dependence relation in $W_{q,NZ}$ contains pairs $(n_q^{div}, n_q^{C_x})$, $(n_q^{div}, n_q^{C_{\bar{x}}\cup\perp})$, $(n_q^{C_x}, n_q^{\otimes})$, $(n_q^{C_{\bar{x}}\cup\perp}, n_q^{\otimes})$, $(n_q^{\otimes}, n_q^{dec})$ and $(n_q^{dec}, n_{q'})$. The role of t_q^{div} is to split $Dass(n_q^{div})$ into disjoint parts: the first one contains records of the form $R(k, C_x)$ and the second part consists of all other remaining records. Tasks $t_q^{C_x}$ and $t_q^{C_{\bar{x}}\cup\perp}$ simply forward their inputs, and task $t_q^{\otimes}$ computes the union of its inputs. Note however that if one of the inputs is empty, the task cannot be executed. Then, task t_q^{dec} deletes one record of the form $R(k, C_x)$. Hence, if $D_q = Dass(n_q)$ is a dataset that contains at least one record of the from $R(k, C_x)$, the execution of all tasks in $W_{q,NZ}$ leaves the system in a configuration with a minimal node $n_{q''}$ labeled by task $t_{q''}$, and with $Dass(n_{q'}) = D_q \setminus R(k, C_x)$

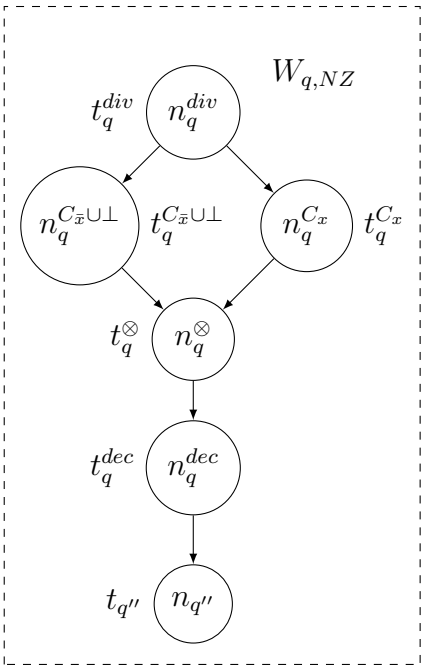


Figure A.3 – Encoding of Non-zero test followed by decrement.

— Let us now detail the contents of $W_{q,Z}$ shown in Figure A.4. This workflow is composed of nodes n_q^{div},$n_q^{C_x\cup\perp}$, n_q^{id}, n_q^{btest}, n_q^{done}, $n_{q'}$ respectively labeled by tasks t_q^{div},$t_q^{C_x\cup\perp}$, t_q^{id}, t_q^{btest}, t_q^{done}, t'_q. The flow relation if given by pairs $(n_q^{div}, n_q^{C_x\cup\perp})$, (n_q^{div}, n_q^{id}), $(n_q^{C_x\cup\perp}, n_q^{btest}), (n_q^{btest}, n_q^{done})$ and (n_q^{id}, n_q^{done}). The role of task t_q^{div} is to project its input dataset on records with $cname = C_x$ or $cname = \perp$, and forward the obtained dataset to node $n_q^{C_x\cup\perp}$. On the other hand, it creates a copy of the input dataset and forwards it to node n_q^{id}. The role of task $t_q^{C_x\cup\perp}$ is to perform a boolean query that returns $\{true\}$ if the dataset contains a record $R(k, C_x)$

and $\{false\}$ otherwise, and forwards the result to node n_q^{btest}. Task t_q^{btest} selects records with value $\{false\}$ (it hence returns an empty dataset as the result of the boolean test was $\{true\}$). Task t_q^{id} forwards its input to node n_q^{done}. Task t_q^{done} receives input datasets from n_q^{btest} and n_q^{id} and forwards the input from n_q^{id} to node $n_{q''}$. One can immediately see that if the dataset input to n_q^{div} contains an occurrence of C_x then one of the inputs to n_q^{done} is empty and hence the workflow deadlocks. Conversely, if this input contains no occurrence of C_x, then this workflows reached a configuration with a single node $n_{q''}$ labeled by task $t_{q''}$, and with the same input dataset as n_q.

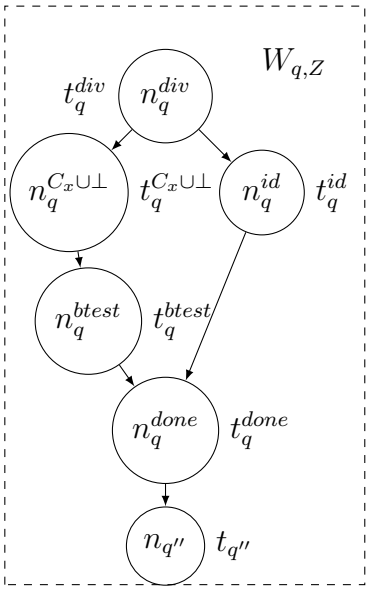


Figure A.4 – Encoding of Zero test followed by state change.

One can see that for every run $\rho = C_0 \dots C_k$ of the two counter machine, where $C_k = (q, v_1, v_2)$ there exists a single non-deadlocked run of the complex workflow, that terminates of configuration $(W, wa, Dass)$ where W consists of a single node n_q labeled by task t_q, and such that $Dass(n_q)$ contains v_1 occurrences of records of the form $R(k, C_1)$ and v_2 occurrences of records of the form $R(k, C_2)$. Hence, a two counter machine terminates in a configuration (q_f, v_1, v_2) iff the only non-deadlocked run of the complex workflow that encodes the two counter machine reaches a final configuration.

■

A.2 Proof of Theorem 2

Theorem 2 : Let CW be a complex workflow, in which tasks do not use SQL difference. Let D_{in} be an input dataset, and $\mathcal{D}_{in}$ be a FO formula. Universal termination of CW on input D_{in} is in $co-2EXPTIME$. Universal termination on inputs that satisfy $\mathcal{D}_{in}$ is undecidable in general. It is in

— $co-2EXPTIME$ (in K, the length of runs) if $\mathcal{D}_{in}$ is in $\forall$FO

— $co-3EXPTIME$ if $\mathcal{D}_{in}$ is $\exists$FO or BSR-FO

— $co-n_{in}$-fold-$EXPTIME$, where $n_{in} = |\mathcal{D}_{in}| + 2^K$ if $\mathcal{D}_{in}$ is in SF-FO.

Proof. Complex workflows terminate iff they have bounded recursive schemes, and if they do not deadlock. This can be verified in $O(|\mathcal{T}_{cx}^2|+|\mathcal{R}|)$ (see proposition 4). If CW is recursion free, checking universal termination of CW can be done by guessing a path. It is hence a non-deterministic process. In the worst case, one may have to explore all symbolic executions $C_0^S \dots C_k^S$ of size at most $3.K_{\mathcal{T}_{cx}}$. If an execution deadlocks, then there is an execution of CW that does not terminate, and we can safely conclude that CW does not terminate universally (for any input). The absence of deadlocks can hence be checked in $EXPTIME$.

If this execution contains a potential deadlock at symbolic configuration C_i^S, then C_i is a configuration from which a particular dataset D must be used as input by a task, and must be empty to cause a deadlock. Before concluding that C_i^S can be a real deadlock, one has to check whether there exists an actual real execution $C_0 \dots C_i$ such that property $D = \emptyset$ holds at C_i. Emptiness of D can be encoded as by a universal FO formula of the form $\psi_i ::= \forall \overrightarrow{X}, rn(\overrightarrow{X}) \notin D$. We can show that the precondition ψ_{i-1} that needs to hold at C_{i-1} in order to obtain an empty dataset D at step C_i are still of the form $D' = \emptyset$ (hence expressible via an universal formula) for some D' (this is the case if the move from C_{i-1} to C_i just adds a field to D' to obtain D) or a FO formula in the universal fragment computed as the weakest precondition $wp[m_i]\psi_i$. Then one can repeat the following steps at each step $k \in i-1, i-2, \dots$ up to C_0^S:

— compute $\psi_k = wp[m_k]\psi_{k+1}$. We know that the weakest precondition can be computed, and that ψ_k is still an universal formula of size in $O(r.|\psi_{k+1}|)$ (see proposition 1).

— Check satisfiability of ψ_k. If the answer is false, then Fail: one cannot satisfy the conditions required to have $D = \emptyset$ at step i, and hence there is no execution with signature $C_0^S \dots C_i^S$ that deadlocks at C_i, the randomly chosen execution is not a witness for deadlock. If the answer is true, continue.

If the algorithm does not stop before step $k = 0$, then the iteration computes a satisfiable formula ψ_0 of size in $O(r^i)$. It remain to show that inputs of the complex workflow meet the conditions in ψ_0.

Let us assume that the universal termination question is considered for a single input dataset D_{in}, one has to check that $D_{in} \models \psi_0$. As ψ_0 is a universal formula i.e. is of the form $\forall \overrightarrow{X}, \varphi_0$, this can be solved in $O(|D_{in}|^{|\psi_0|})$. If the answer is true then we have found preconditions that are satisfied by D_{in} and that are sufficient to obtain an empty dataset at configuration C_i in a run $C_0 \dots C_i$ that has signature $C_0^S \dots C_i^S$, i.e. $C_0^S \dots C_i^S$ witnesses the existence of a deadlock. Overall, one has to solve up to $i < 3.K_{\mathcal{T}_{cx}}$ satisfiability problems for universal FO formulas $\psi_{i-1}, \psi_{i-2} \dots \psi_1$ of size smaller than r^i, and a model checking problem for input D_{in} with a cost in $O(|D_{in}|^{r^i})$. The satisfiability problems are $NEXPTIME$ in the size of the formula [Lew80] and hence checking satisfiability of $\psi_{i-1}, \dots, \psi_0$ has a complexity that is doubly exponential in $K_{\mathcal{T}_{cx}}$. Considering that data fields are encoded with c bits of information, D_{in} is a dataset of size in $O(2^{r.c})$. Hence, the overall complexity to check that $C_0^S \dots C_i^S$ is a witness path that deadlocks in $2 - EXPTIME$.

Conversely, if the universal termination question is considered for several input datasets described with a FO formula $\mathcal{D}_{in}$, one has to check that no contradiction arises when requiring the existence of input D_{in} that satisfies both $\mathcal{D}_{in}$ and ψ_0. This can be done by checking satisfiability of the conjunction $\mathcal{D}_{in} \wedge \psi_0$. The formula is of size $|\mathcal{D}_{in}| + |\psi_0|$, and one can consider variables in $\mathcal{D}_{in}$ and ψ_0 to be disjoint. Standard equivalence rules (miniscoping rules, see def.14) allow rewriting this conjunction into an equivalent formula in prenex normal form. If $\mathcal{D}_{in}$ is in $\forall$FO, then $\mathcal{D}_{in} \wedge \psi_0$ is in $\forall$FO and we have a co-2EXPTIME procedure to verify its satisfiability. For fragments ($\exists$FO, BSR-FO), $\mathcal{D}_{in} \wedge \psi_0$ fall in the class of BSR-FO or SF-FO formulas, but with three alternations, with a NEXPTIME complexity of satisfiability, yielding a triple exponential complexity. For $\mathcal{D}_{in}$ in SF-FO, the complexity of the last step is n_{in}-fold exponential in the size of $\mathcal{D}_{in}$ plus the size of ψ_0. As for the unique input case, if $\mathcal{D}_{in} \wedge \psi_0$ is satisfiable, then $C_0^S \dots C_i^S$ witnesses existence of a non-terminating execution. Hence, one can witness existence of a non-terminating run in $O(K_{\mathcal{T}_{cx}}.2^{r^{K_{\mathcal{T}_{cx}}}} + C_{in})$ where C_{in} is the cost required to check satisfiability of $\mathcal{D}_{in} \wedge \psi_0$. Last if $\mathcal{D}_{in}$ is specified in an undecidable fragment of FO, then one cannot conclude whether there exists a legal input that satisfies $\mathcal{D}_{in}$ and ψ_0. ■

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : **Workflows centrés sur les données pour l'application Crowdsourcing**

Mot clés : Crowdsourcing, workflows centrés sur les données, assurance qualité

Resumé : Le crowdsourcing utilise l'intelligence humaine pour résoudre des tâches difficiles à réaliser par des machines. Les plateformes de crowdsourcing existantes permettent de réaliser des lots de micro-tâches très simples. Cependant, de nombreux processus sont des tâches complexes, qui nécessitent d'enchaîner la collecte de données, des prétraitements, de l'analyse de données, de la synthèse, etc. Dans cette thèse, nous étudions comment spécifier ces tâches complexes, pour les faire réaliser par des plateformes de crowdsourcing. Nous proposons tout d'abord le modèle des *workflows complexes* qui fournit des constructions de haut niveau pour décrire une tâche complexe comme une orchestrations d'un ensemble de tâches simples. Nous fournissons des algorithmes permettant de vérifier la terminaison et la correction de ces workflows pour un sous-ensemble du langage (ces questions étant indécidables dans le cas général). Un des inconvénients du crowdsourcing est le fait que de mauvaises réponses peuvent être produites par les agents humains. Pour pallier à ce problème, il est habituel de répliquer les tâches, puis d'aggréger les résultats pour fiabiliser une réponse finale. La réplication augmente la qualité des données, mais elle est coûteuse. Nous proposons des techniques d'agrégation de résultats dans lesquelles l'aggrégation est réalisée à partir d'algorithmes d'Expectation Maximization, et la réplication est faite à la demande en tenant compte de la confiance estimée sur les agrégats. Les résultat expérimentaux montrent que ces techniques permettent de regrouper les réponses tout en obtenant un bon compromis coût-fiabilité pour des lots de micro-tâches, mais aussi pour des tâches complexes.

Title : **Data Centric Workflows for Crowdsourcing Application**

Keywords : Crowdsourcing, Data-centric workflows, Quality assurance.

Abstract : Crowdsourcing uses human intelligence to solve tasks which are still difficult for machines. Tasks at existing crowdsourcing platform are batches of relatively simple micro-tasks. However, real-world problems are often more difficult than micro-tasks. They require data collection, organization, pre-processing, analysis, and synthesis of results. In this thesis, we study how to specify complex crowdsourcing tasks and realize them with the help of existing crowdsourcing platforms. The first contribution of this thesis is a *complex workflows* model that provides high-level constructs to describe a complex task through orchestration of simpler tasks. We provide algorithms to check termination and correctness of a complex workflow for a subset of the language (these questions are undecidable in the general case). A well-known drawback of crowdsourcing is that human answers might be wrong. To leverage this problem, crowdsourcing platforms replicate tasks, and forge a final trusted answer out of the produced results. Replication increases quality of data, but it is costly. The second contribution of this thesis is a set of aggregation techniques where merging of answers is realized using Expectation Maximization, and replication of tasks is performed online after considering the confidence estimated for aggregated data. Experimental results show that these techniques allow to aggregate the returned answers while achieving a good trade-off between cost and data quality, both for the realization of a batches of micro-tasks, and of complex workflow.