

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Language-Based Differential Privacy with Accuracy Estimations and Sensitivity Analyses

ELISABET LOBO-VESGA



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2023

Language-Based Differential Privacy with Accuracy Estimations
and Sensitivity Analyses

Elisabet Lobo-Vesga

© Elisabet Lobo-Vesga, 2023

ISBN 978-91-7905-811-1

Doktorsavhandlingar vid Chalmers tekniska högskola

Ny serie nr 5277

ISSN 0346-718X

Department of Computer Science & Engineering

Chalmers University of Technology

SE-412 96 Gothenburg, Sweden

Telephone +46 (0)31-772 1000

Printed at Reproservice, Chalmers University of Technology
Gothenburg, Sweden, 2023

Language-Based Differential Privacy with Accuracy Estimations and Sensitivity Analyses

ELISABET LOBO-VESGA

Department of Computer Science & Engineering
Chalmers University of Technology

Abstract

This thesis focuses on the development of programming frameworks to enforce, by construction, desirable properties of software systems. Particularly, we are interested in enforcing differential privacy—a mathematical notion of data privacy—while statically reasoning about the accuracy of computations, along with deriving the sensitivity of arbitrary functions to further strengthen the expressiveness of these systems. To this end, we first introduce DPella, a programming framework for differentially-private queries that allows reasoning about the privacy and accuracy of data analyses. DPella provides a novel component that statically tracks the accuracy of different queries. This component leverages taint analysis to infer statistical independence of the different noises that were added to ensure the privacy of the overall computation. As a result, DPella allows analysts to implement privacy-preserving queries and adjust the privacy parameters to meet accuracy targets, or vice-versa.

In the context of differentially-private systems, the sensitivity of a function determines the amount of noise needed to achieve a desired level of privacy. However, establishing the sensitivity of arbitrary functions is non-trivial. Consequently, systems such as DPella provided a limited set of functions—whose sensitivity is known—to apply over sensitive data; thus hindering the expressiveness of the language. To overcome this limitation we propose a new approach to derive proofs of sensitivity in programming languages with support for polymorphism. Our approach enriches base types with information about the metric relation between values and applies parametricity to derive proof of a function’s sensitivity. These ideas are formalized in a sound calculus and implemented as a Haskell library called SPAR, enabling programmers to prove the sensitivity of their functions through type-checking alone.

Overall, this thesis contributes to the development of expressive programming frameworks for data analysis with privacy and accuracy guarantees. The proposed approaches are feasible and effective, as demonstrated through the implementation of DPella and SPAR.

List of publications

Appended publications

This dissertation is for the degree of Doctor of Philosophy, and includes reprints of the following papers:

Paper A “A Programming Language for Data Privacy with Accuracy Estimations”

Elisabet Lobo-Vesga, Alejandro Russo, and Marco Gaboardi
ACM Transactions on Programming Languages and Systems (TOPLAS), 2021.

Observations. This paper expands upon the results of our published work:

“A Programming Framework for Differential Privacy with Accuracy Concentration Bounds”

Elisabet Lobo-Vesga, Alejandro Russo, and Marco Gaboardi
Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), 2020.

Paper B “Sensitivity by Parametricity: Simple Sensitivity Proofs for Differential Privacy”

Elisabet Lobo-Vesga, Alejandro Russo, and Marco Gaboardi
Manuscript under submission.

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to those who contributed and accompany me throughout my doctoral journey. First, I would like to express my deepest gratitude to Alejandro Russo, my supervisor. His patient guidance, inspiring enthusiasm, and constant encouragement have been instrumental in shaping my research and have played a critical role in my academic and personal growth. He has consistently challenged me to think critically and pushed me to improve my skills, including my Street Fighter II abilities – I remain determined to win a match someday! I am truly grateful for his mentorship and advice throughout this whole process. I would also like to express my appreciation to my co-supervisor and collaborator, Marco Gaboardi, for his detailed lessons on probability and type theory. His expertise and guidance have been invaluable, I am grateful for the opportunity to have worked with him.

To my friends and colleagues, I extend my sincere thanks for their fellowship and insightful conversations. Their encouragement has been a source of inspiration and motivation for me. I am forever grateful for the lifelong friendships that we have formed while sharing this experience; it truly has been an honor.

I owe a debt of gratitude to my family who has been my constant support system. Their love, prayers, and encouragement have been a driving force behind my academic endeavors. I am especially grateful to my parents, who instilled in me a love of learning and have supported me every step of the way; and to my brother, the person whom I admire the most and who has paved the way for my success. Finally, words cannot express how thankful I am for my partner Alejandro Gómez; for always believing in me, thanks.

To each and every person who has been part of this experience, I am forever grateful. Your input has made this journey possible, and I am thankful for your presence in my life.

March 1st, 2023

Contents

Abstract	iii
List of publications	v
Acknowledgments	vii
I Introduction	1
I.1 Privacy protection in context	2
I.2 Differential privacy	5
I.2.1 Properties	6
I.2.2 Models and tools	6
I.2.3 Challenges	7
I.3 Statement of contribution	8
I.3.1 Addressing challenge 1	8
I.3.2 Addressing challenge 2	9
<hr/>	
Accuracy	
A A Programming Language for Data Privacy with Accuracy Estimations	15
A.1 Introduction	15
A.2 Background	19
A.3 DPella by example	21
A.3.1 Basic aggregations	21
A.3.2 Cumulative Distribution Function	24
A.4 Privacy	28
A.4.1 Components of the API	28
A.4.2 Transformations	30
A.4.3 Partition	30
A.4.4 Aggregations	32
A.4.5 Privacy budget and execution of queries	32
A.4.6 Implementation	33
A.5 Accuracy	33
A.5.1 Accuracy calculations	33
A.5.2 Implementation	37
A.5.3 Accuracy of Gaussian mechanism	40
A.6 Case studies	42
A.6.1 DPella expressiveness	43
A.6.2 Privacy and accuracy trade-off analysis	46
A.6.3 K-way marginal queries on synthetic data	48
A.7 Testing accuracy	50
A.8 API generalization	53

A.8.1	Implementation and Accuracy estimations	54
A.9	Limitations & Extensions	56
A.10	Related work	59
A.11	Conclusions	61
	Bibliography	63

Sensitivity

B	Sensitivity by Parametricity: Simple Sensitivity Proofs for Differential Privacy	69
B.1	Introduction	69
B.1.1	Motivating examples	71
B.2	λ_{SPAR} : a calculus for distance tracking	73
B.2.1	Syntax	73
B.2.2	Operational semantics	77
B.3	Formal guarantees	77
B.3.1	Sensitivity by Parametricity	80
B.4	λ_{SPAR} as a library	82
B.4.1	Vectors	83
B.4.2	Currying	85
B.4.3	Sorting	86
B.4.4	Beyond sensitivity	86
B.5	Implementation	87
B.5.1	Equality for type-level natural numbers	87
B.5.2	SPAR as an embedded DSL	89
B.5.3	Executing functions	90
B.6	Discussion	91
B.7	Related work	92
B.8	Conclusions	94
Appendix		95
B.A	SPAR's complete syntax	95
B.B	Typing system	96
B.C	Semantics	99
B.C.1	Operational	99
B.C.2	Distance and length interpretation	100
B.D	Logical relations	100
B.D.1	Definitions	101
B.E	Proof of metric preservation and accompanying lemmas	102
B.E.1	Fundamental lemma of logical relations	112
	Bibliography	145



Introduction

Constantly sharing personal information has been integrated into our daily routines. From our home devices, online interactions, and the services we use; to more explicit disclosures such as filling out forms and answering surveys, our data is being collected, stored, sold, and processed by a wide range of agents. These agents (e.g., research institutions, government agencies, and businesses) rely on collected data to improve their services, understand populations, tailor policies, and make informed decisions. Consequently, data processing is at the backbone of our society and has the potential to impact our communities and lives positively. It is then desirable to share our information with such agents for personal and societal gains. However, the information provided often contains confidential and sensitive details about ourselves that we expect to remain private and accessible only to those trusted parties; unfortunately, this has not always been the case.

The mishandling of sensitive data has become commonplace among companies and public institutions [17, 14, 21, 27]. As a result, many laws, regulations, and agreements [6, 4, 16, 2] have been put in place recognizing the importance of protecting individuals' privacy and mitigating the occurrence of privacy breaches. Improper disclosure of the information is severely penalized with fines which might put some companies out of business or heavily affect their reputation and competitiveness [28, 1, 19, 3]. To make matters worst, when privacy breaches occur, they are irreversible and have lingering consequences on those affected. These incidents perpetuate distrust between the individuals and the agents interested in their data, deterring the public from sharing their information in the future [10, 7, 11]. The vast implications of privacy breaches then severely limit the potential usage of individuals' data and its availability altogether.

It is in everyone's interest to avoid privacy breaches, but ensuring data privacy is a complex problem. Companies, researchers, and policymakers have searched for robust and concrete ways to define, ensure, and regulate data privacy. Decades of trial and error have made it evident that data privacy cannot be achieved with a few hacks or as an afterthought. Instead, it must be a fundamental approach that can withstand technological changes and unforeseen risks while being feasible for today's needs.

Are our requirements for data privacy utopian? Should data analysis be halted or reduced to preserve individuals' privacy? Fortunately, that is not the case; various privacy-preserving techniques are available that allow us to perform statisti-

cal data analyses while guaranteeing the privacy of individual participants. One such approach is differential privacy [13], a mathematical and quantifiable definition of privacy that has gained popularity for its provable guarantees and applicability. Nevertheless, as the problem of privacy is broad and intricate, it is essential to clarify the domain in which differential privacy is applicable and the drawbacks that it might have. Concretely, this dissertation explores some challenges concerning the deployment and usability of differential privacy and addresses them in the context of programming languages.

Before diving into the opportunities and challenges of differential privacy, it is important to explore the context of privacy protection and its threats. Following the reader can find a brief description of some well-known and relevant techniques used by data analysts and privacy practitioners in their daily tasks. This primer will serve as an introduction to the field of statistical data privacy and as a motivation for the usage of the study and application of differential privacy.

I.1 Privacy protection in context

Data anonymization or de-identification. Is the process of removing personally identifying information (PII) from datasets so that the remaining information cannot be linked to specific individuals. In practice, these techniques require data owners to pre-process datasets by purging *explicit identifiability* information such as names and government-issued IDs; as well as *potentially identifiability* information such as IP addresses or next of kin. The remaining data presents a best-of-both-words scenario in which unscrupulous actors will not be able to identify the people providing the information, and honest analysts will have useful data to perform their studies.

The promise of yielding *useful and privacy-preserving* results has positioned anonymization techniques as the *de-facto* approach among practitioners storing, sharing, and processing sensitive data. This sense of assurance is further reinforced by regulatory agents and globally common statutes in which anonymization is considered *sufficient* to protect individuals' privacy [26]. Despise its apparent robustness, data breaches still occur in the presence of anonymized data.

The weakness of anonymization techniques is their incapacity to account for data's multiple degrees of identifiability. While PII's are indeed attributes an adversary can use to identify an individual, the same result can be achieved by combining attributes that do not classify as personally identifiable. For instance, Sweeney [31] demonstrated that the combination of ZIP code, birth date, and sex are unique to 87% of the American population. Furthermore, when considering other available sources of information, the probability of uniquely identifying individuals is increased by cross-referencing with the anonymized data. It is then clear that data anonymization is susceptible to privacy attacks and cannot always fulfill its promise of providing useful and privacy-preserving results.

Privacy attacks on anonymized data aim to reverse the process of anonymization. Attackers can exploit the aforementioned vulnerabilities by associating anonymized records with non-anonymized information from different datasets, this tech-

nique is known as a linkage attack. Using non-anonymous data as background knowledge, attackers are capable to trace back individuals (known as re-identification attacks) or recover large portions of the original dataset (known as reconstruction attacks). Even though these attacks might seem difficult to perform and unlikely to succeed, concrete instances of such attacks abound. Consequently, I present two infamous cases in which sophisticated data administrators overestimated anonymization guarantees and compromised the privacy of hundreds of people.

- **AOL Searcher No. 4417749:** To provide useful data for academic research, AOL released a dataset of search queries performed by its users. The company anonymized said data by replacing user IDs with random numbers and removing IP addresses. The combination of searches performed by a user—whose identity was hidden behind an associated random number—were naively considered non-identifiable attributes of that user. Later on, New York Times journalists Barbaro and Zeller, prove this assumption to be false [8]. In the article, the authors showcase how a set of searches can reveal particular characteristics of the users. Concretely, they re-identified and presented user No. 4417749, a 62-year-old widow searching for "numb fingers", "60 single men", "dog that urinates on everything", "homes sold in shadow lake subdivision gwinnett county georgia.", and "landscapers in Lilburn, Ga.". When notified about the vulnerabilities, AOL removed the dataset and apologized for its publication, but, as pointed out by the authors, the data was already copied and distributed on other sites; thus leaving AOL users' permanently exposed.
- **Netflix competition:** Netflix released a dataset containing movie ratings provided by their users as part of training data for a competition to improve their recommendation algorithm. To anonymize the dataset, user IDs were replaced, several ratings were randomly altered and dates were modified. Despite their efforts, Narayanan and Shmatikov [25] demonstrated that more 80% of the users were identifiable by knowing the time and rating of only three movies. By using publicly available ratings from the Internet Movie Database (IMDB) as background knowledge, the authors were able to re-identify common users across the datasets, in addition to learning other potentially sensitive information such as users' apparent political preferences.

These examples exhibit the prevalence of using anonymization for privacy preservation among practitioners, but more importantly, they demonstrate the theoretical and practical limitations of this technique, casting substantial doubts about anonymization's power for ensuring privacy.

Summary statistics. A common refrain among data analysts is to "aggregate" data to make it safe to share and release. The idea behind this approach is that it provides a hide-in-the-bunch effect where individuals are not likely to be singled out. Intuitively, this simple approach fulfills the promise of protecting individuals' privacy, after all, how can an attacker know my specific salary if all that is shared is the average income of people in my area? As it turns out, this intuition is full of risks and potential mistakes.

Region	Age	Sex	Count
A	20 - 29	F	-
A	30 - 39	F	20
A	40 - 49	F	9
A	50 - 59	F	17
A	20 - 59	F	49

Table I.1: Summary female population in region A

Straight-forward attacks can be foreseen under the presence of outliers or when the population is not big enough to "hide" data points. In fact, the field of statistical disclosure control [29] aroused from the need to protect information on tabular and aggregated data. Consequently, statistical organizations have devised various methods to mitigate these attacks, among them, the *threshold rule* stands as the most commonly used [5]. The threshold rule consists on requiring a minimum number of respondents (per categorization) in order to provide the aggregated results. For instance, applying a threshold of 5 would mean that at least 5 individuals must share the same combination of age, sex, and region of residence in order to provide any insights about a population with this categorization. Although the threshold rule is simple to implement and seemingly efficient to prevent issues with identifying eccentric data points; the privacy guarantees are broken when the aggregated statistics are reversible and the releases are accumulated through time.

Consider the aggregated data in Table I.1 containing the summary statistics of the female population in a certain region. Here the population of females is aggregated within age ranges, additionally, the total population of females (known as a marginal statistic) is provided. With this information, we can easily identify that number of females between the ages of 20-29 is $49 - 20 - 9 - 17 = 3$. Even though this example presents an obvious scenario, reversing aggregations across many dimensions when marginal statistics are included is a well-known and common problem [9].

Releasing marginal statistics jeopardizes the privacy guarantees provided by summary statistics, however, privacy-by-aggregation's vulnerabilities exist beyond marginal summaries. When aggregated data is produced over time, attackers are provided with additional information that can be used to compare and infer sensitive information. Say our previous example was produced for January, in which **{Region:A, Age:40-49, Sex: F}** = 9; if the results of the following month are **{Region:A, Age:40-49, Sex: F}** = 10, it is easy to notice that one person has been added thus, violating the threshold rule since less than 5 individuals are represented in the difference between both counts. To make matters worst, if an attacker has previous knowledge of Alice moving to region A in this period, they can infer that Alice is between 40 to 49 years old.

The main problem with privacy-via-aggregation is that all methods of numerical aggregation can be used to reconstruct the original data. This phenomenon was called the *Fundamental Law of Information Recovery* by Dwork and Roth [13] stating that "overly accurate answers to too many questions will destroy privacy in a spec-

tacular way". Intuitively, the more statistics generated from a single set of data, the greater the chance of reconstructing the original data from those statistics; this is simply because each release decreases the possibilities for the data that could have produced those statistics. This is why prominent data managers including the U.S. Census Bureau [24], Google [15], and Apple [18], have shifted their interest to more robust tools for releasing privacy-preserving statistics such as differential privacy.

I.2 Differential privacy

Differential privacy [13] is a formal mathematical definition of privacy in aggregate statistics (e.g., averages and histograms) and machine learning analysis (e.g., k-means and stochastic gradient descent). This formal framework has gained increasing popularity during the past decades as its core mechanisms are a variant of the classic *randomized response* [32], protecting individuals' privacy with formal guarantees of *plausible deniability*—i.e., when performing a statistical analysis over a dataset, any participant can deny the presence of their information in the input data. Accordingly, differential privacy ensures that anyone observing the result of a differentially-private computation will likely make the same inferences about an individual, whether or not their information is included as input for the analysis. Furthermore, differential privacy specifies mathematical assurance for privacy protection against various privacy attacks such as re-identification, reconstruction, and differencing attacks.

The success of differential privacy lies in the fact that it identifies algorithms as the primary culprits for data breaches. Under differential privacy, data is not anonymized, as we have seen that this technique is susceptible to linkage attacks [30, 25, 8, 12]. Additionally, differential privacy does not rely on the potential privacy of aggregated statistical results, as this approach is susceptible to reconstruction and membership attacks [20, 12]. Instead, differential privacy focuses on how the algorithms at hand can influence the relationship between the input (possibly sensitive) data and the outcome of the computations. In this sense, differential privacy is not a single tool or implementation but a *criterion* or *property* that many algorithms for accessing sensitive personal data are devised to satisfy.

Intuitively, an algorithm (often referred to as query or analysis) is said to satisfy differential privacy when it returns statically indistinguishable outputs when given two datasets differing in the data of a single individual. In order to fulfill this condition, differentially-private algorithms add calibrated noise to their result to mask the absence, inclusion, or modification of someone's information in the input dataset. The strength of differential privacy's guarantees can be tuned via the *privacy parameter* ϵ ¹. This parameter is commonly referred to as the *privacy loss* as it can be interpreted as the additional risk a participant is exposed to by partaking in a specific data analysis. Consequently, the value of ϵ directly influences the noise in a computation's result to ensure privacy. As ϵ decreases, the strongest the privacy

¹In its general form, differential privacy is parametrized by (ϵ, δ) , with ϵ bounding the total privacy loss and δ referring to a failure probability of the DP guarantees.

guarantees are; however, this comes at the cost of adding more noise, thus impacting the results' accuracy.

At the core of every differentially-private algorithm lies the noise calibration mechanism. Noise calibration is crucial to provide both useful and private results. While the ϵ parameter quantifies the desired level of privacy, we also need to consider how susceptible the algorithm is to disclose an individual's information when the dataset changes. The quantification of how much an operation's result changes relative to its inputs is known as *sensitivity*. Together, ϵ and the algorithm's sensitivity provide us with enough information to determine how much noise is needed to achieve differential privacy.

I.2.1 Properties

The rigorous mathematical guarantees provided by differential privacy yield several practical benefits for its users [13]:

Composability. Differential privacy features beneficial compositional properties allowing analysts to create complex analyses using basic ones. The principle of *sequential composition* is one of the most basic ones stating that if a family of algorithms \mathcal{A}_i satisfy ϵ_i differential privacy, then executing a sequence of the algorithms satisfies $\sum_i \epsilon_i$ -differential privacy. The principle of *advanced composition* can produce tighter limits on their total privacy loss when considering iterative algorithms.

Provable guarantees. Differential privacy is the only existing approach providing provable privacy guarantees for successive data releases.

Transparency. Differentially private algorithms and their parameters are not secrets to be protected. Opposite to traditional de-identification tools, knowing the extent to which data has been transformed does not threaten the differential privacy guarantees. This transparency boosts reproducibility, accountability, and public trust in the process of data analysis.

Post-processing resilience. Differential privacy guarantees that any subsequent processing of data releases does not increase the risk of privacy violation for individuals.

Group privacy. While differential privacy is commonly used to protect privacy at the individual level, it has been shown that its guarantees also translate to (weaker) protection for groups of individuals. Concretely, an algorithm satisfying ϵ -differential privacy for individuals also provides $k\epsilon$ -differential privacy for groups of size k .

I.2.2 Models and tools

Since differential privacy is a mathematical property, there are multiple ways in which we can design algorithms to fulfill it. Depending on whether data collectors

are trusted, differentially private algorithms can be executed centrally or locally. In a centralized setting, the individuals transmit their raw data to trusted parties; it is assumed that these entities will safely store the data and correctly use differential privacy to access the information. In contrast, in a local setting, data collectors are not trusted; therefore, each participant will perturb their response before sharing; hence sensitive information is never stored in one location.

The local model seemingly provides the ideal scenario where privacy is guaranteed, and security is boosted by avoiding creating honeypots for hackers. However, several studies have shown that these algorithms do not perform as accurately as those in the centralized model for the same level of privacy. Consequently, most differential privacy tools—including those introduced in this dissertation—are based on the centralized model.

Most frameworks for differential privacy are based on the same principle: they provide a set of fundamental private analyses, which the analysts can use as building blocks to create more complex algorithms. This approach relies heavily on the compositional property as this principle will determine the final privacy guarantees of the combined analyses.

I.2.3 Challenges

Privacy-accuracy reasoning

Composability is a fundamental property for developing programming tools for differential privacy. When combining building blocks, these tools ensure that the total privacy loss of the resulting analysis does not exceed the desired privacy level. This characteristic facilitates reasoning about an analysis' total privacy loss as a budget that is distributed and spent through the algorithms' pieces.

Strongly connected to privacy is the concept of accuracy. Analysts might be interested in controlling their algorithms' privacy and accuracy. One could argue that privacy is a concern solely for the individuals (and data holders), while accuracy is a concern exclusively for the analysts (and those interested in the statistical analyses). Unfortunately, reasoning about accuracy is less compositional than reasoning about privacy. Determining the accuracy of arbitrary user-defined algorithms is complex as it depends on the specific task at hand and the specific error measurement. In the literature, most of the standard algorithms for differentially-private analyses are provided with accuracy estimations (in the form of confidence intervals or error bounds); however, the accuracy of their combination is addressed on a case-by-case basis. As a result, most programming frameworks for differential privacy do not offer any support for tracking, reasoning, and adjusting the accuracy of the algorithms; the crucial task of predicting accuracy is left to the analysts.

Proof of sensitivity for user-defined functions

Noise calibration is at the backbone of every differentially private algorithm. To sample the adequate noise required to satisfy differential privacy, we need to consider the desired privacy level (ϵ) and the sensitivity of the algorithm at hand (a measurement of how volatile it is to changes in its inputs). Unfortunately, determining

the sensitivity of arbitrary operations can be challenging. For this reason, most programming tools for differential privacy do not support the definition of arbitrary operations. Instead, they are equipped with predefined operations whose sensitivity is known, avoiding sensitivity calculations altogether. However, even though predefined operations have allowed for many exciting analyses, it severely constrains the kind of computations we can perform on the datasets, thus limiting access to valuable information.

Several programming frameworks have been proposed to compute the sensitivity of user-defined operations. The typical approach to statically computing the sensitivity of a program consists of providing a language with a type system enriched with sensitivity annotations. Then, when combining the provided primitives, the type system will keep track of the program's global sensitivity. Unfortunately, most of these frameworks are never fully deployed because they often rely on advanced features not available in mainstream programming languages, thus requiring creating full-stack languages from scratch. Moreover, those frameworks that manage to create a functioning prototype lack acceptance by data analysts since the tools are based on niche programming devices (e.g., linear and modal types) unknown outside academic circles.

I.3 Statement of contribution

This dissertation encompasses a series of works proposing several programming techniques to help non-experts write differentially private algorithms and reason about the different components of these algorithms. With the deployment of such techniques, we expect to equip data analysts with tools where i) they can create data analysis satisfying differential privacy by construction, ii) they can reason about the privacy-accuracy trade-offs before execution and, iii) they are not limited to a set of predefined algorithms to create their own.

At a high level, the contributions of this dissertation can be grouped into two categories, each of them tackling one of the challenges listed above:

I.3.1 Addressing challenge 1

We created DPella, a programming framework for differentially-private algorithms that allows data analysts to reason compositionally about privacy-accuracy trade-offs at compile time. DPella's main novelty is that it exemplifies how programming frameworks can internalize the use of probabilistic bounds for composing different confidence intervals or error bounds in an automated way. DPella leverages taint analysis to detect statistical independence of the noise added by its different primitives; this information is then used to achieve better error estimates. Finally, since DPella's analysis is data-independent, it showcases how mainstream statically-typed languages can be used to perform differential privacy analysis as part of their type-checking process without relying on any runtime execution or information.

These results are recorded in our work "A Programming Language for Data Privacy with Accuracy Estimations." *In 2021 ACM Transactions on Programming Lan-*

guages and Systems (TOPLAS) [23] which in turn is an extension of our previous work "A Programming Framework for Differential Privacy with Accuracy Concentration Bounds." *In 2020 IEEE Symposium on Security and Privacy (SP)* [22]. The former is the only one included in this dissertation as it encompasses both results; the main differences between both works are highlighted in a subsequent section A.1.

I.3.2 Addressing challenge 2

We proposed a sound calculus (λ_{SPAR}) for statically determining the sensitivity of user-defined programs while avoiding using linear and relational refinement types. Our approach relies on a novel use of parametricity—a well-known abstract uniformity property enjoyed by polymorphic functions—together with type constraints and type-level naturals to verify a program’s sensitivity by simply type-checking. Its simplicity facilitates embedding λ_{SPAR} into mainstream richly-typed programming languages.

We introduced SPAR, a concrete implementation of λ_{SPAR} as a library for the Haskell programming language. The library SPARis implemented as an embedded domain-specific language which allows us to leverage Haskell’s advanced type inference to provide some support for sensitivity inference via type error—a feature that, to our knowledge, has not been explored before. Finally, we complemented our findings with the implementation of classic examples (such as summing, mapping, and sorting elements of a vector) to demonstrate how SPARcan be used to prove user-defined programs’ sensitivity. The main result of this work opens the door to integrating procedures for automatically proving the sensitivity of user-defined analyses into the programming workflow, e.g., by using SPAR’s sensitivity proofs as an input to other Haskell-based DP frameworks.

Bibliography

- [1] IBM: Cost of a data breach report. *Network Security*, 2022(8):4, 2022.
- [2] AB-375. California consumer privacy act (CCPA). 34, 2018.
- [3] A. Acquisto, A. Friedman, and R. Telang. Is there a cost to privacy breaches? an event study. In *ICIS 2006 Proceedings - Twenty Seventh International Conference on Information Systems*, pages 1563–1580, 2006.
- [4] P. Act. Family educational rights and privacy act (FERPA). 2014.
- [5] L. Arbuckle. Aggregated data provides a false sense of security. <https://iapp.org/news/a/aggregated-data-provides-a-false-sense-of-security/>, April 2020. [Online; posted 27-April-2020].
- [6] U. G. Assembly et al. Universal declaration of human rights. *UN General Assembly*, 302(2):14–25, 1948.
- [7] G. Bansal, F. Zahedi, and D. Gefen. The impact of personal dispositions on information sensitivity, privacy concern and trust in disclosing health information online. *Decision Support Systems*, 49(2):138–150, 2010.
- [8] M. Barbaro and T. Zeller Jr. A face is exposed for AOL searcher no. 4417749. <https://www.nytimes.com/2006/08/09/technology/09aol.html>, August 2006. [Online; posted 09-August-2006].
- [9] L. Buzzigoli and A. Giusti. From marginals to array structure with the shuttle algorithm. *Journal of Symbolic Data Analysis*, 4(1):1–14, 2006.
- [10] H. Choi, J. Park, and Y. Jung. The role of privacy fatigue in online privacy behavior. *Computers in Human Behavior*, 81:42–51, 2018.
- [11] M. J. Culnan and P. K. Armstrong. Information privacy concerns, procedural fairness, and impersonal trust: An empirical investigation. *Organization Science*, 10(1):104–115, 1999.
- [12] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210, 2003.
- [13] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.
- [14] H. Elizabeth A. and N. Perloth. For target, the breach numbers grow. *The New York Times (January 10)*, 2014.
- [15] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

- [16] European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), 2016.
- [17] J. Finkle and D. Skariachan. Target cyber breach hits 40 million payment cards at holiday peak. *Reuters (December 18)*, 2013.
- [18] A. Greenberg. Apple’s ‘differential privacy’ is about collecting your data—but not *Your* data. <https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/>, June 2016. [Online; posted 13-June-2016].
- [19] M. Henriquez. \$4.35 million – the average cost of a data breach. *Security*, 59(10):7, 2022.
- [20] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig. Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays. *PLoS genetics*, 4(8):e1000167, 2008.
- [21] N. Horton and A. DeSimone. Sony’s nightmare before christmas: The 2014 north korean cyber attack on sony and lessons for us government actions in cyberspace. Technical report, JHUAPL Laurel United States, 2018.
- [22] E. Lobo-Vesga, A. Russo, and M. Gaboardi. A programming framework for differential privacy with accuracy concentration bounds. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 411–428. IEEE, 2020.
- [23] E. Lobo-Vesga, A. Russo, and M. Gaboardi. A programming language for data privacy with accuracy estimations. *ACM Trans. Program. Lang. Syst.*, 43(2), June 2021.
- [24] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: From theory to practice on the map. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 277–286.
- [25] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.
- [26] P. Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA L. Rev.*, 57:1701, 2009.
- [27] S. A. O’Brien. Giant equifax data breach: 143 million people could be affected. *CNN Tech*, 8, 2017.
- [28] J. Ruohonen and K. Hjerpe. The gdpr enforcement fines at glance. *CoRR*, abs/2011.00946, 2020.

- [29] C. Skinner. Chapter 15 - Statistical Disclosure Control for Survey Data. In C. Rao, editor, *Handbook of Statistics*, volume 29 of *Handbook of Statistics*, pages 381–396. Elsevier, 2009.
- [30] L. Sweeney. Weaving technology and policy together to maintain confidentiality. *The Journal of Law, Medicine & Ethics*, 25(2-3):98–110, 1997.
- [31] L. Sweeney. Uniqueness of simple demographics in the us population. *LIDAP-WP4, 2000*, 2000.
- [32] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.



A Programming Language for Data Privacy with Accuracy Estimations

Abstract. Differential privacy offers a formal framework for reasoning about the privacy and accuracy of computations on private data. It also offers a rich set of building blocks for constructing private data analyses. When carefully calibrated, these analyses simultaneously guarantee the privacy of the individuals contributing their data, and the accuracy of the data analysis results, inferring useful properties about the population. The compositional nature of differential privacy has motivated the design and implementation of several programming languages to ease the implementation of differentially private analyses. Even though these programming languages provide support for reasoning about privacy, most of them disregard reasoning about the accuracy of data analyses. To overcome this limitation, we present DPella, a programming framework providing data analysts with support for reasoning about privacy, accuracy, and their trade-offs. The distinguishing feature of DPella is a novel component that statically tracks the accuracy of different data analyses. In order to provide tight accuracy estimations, this component leverages taint analysis for automatically inferring *statistical independence* of the different noise quantities added for guaranteeing privacy. We evaluate our approach by implementing several classical queries from the literature and showing how data analysts can calibrate the privacy parameters to meet the accuracy requirements, and vice-versa.

A.1 Introduction

Differential privacy (DP) [18] is emerging as a viable solution to release statistical information about the population without compromising data subjects' privacy. A standard way to achieve DP is by adding some statistical noise to the result of a data analysis. If the noise is carefully calibrated, it provides *privacy* protection for the individuals contributing their data. At the same time, it enables the inference of *accurate* information about the population from which the data are drawn. Thanks to its quantitative formulation, quantifying privacy by means of the parameters ϵ and δ , DP provides a mathematical framework for rigorously reasoning about the *privacy-accuracy trade-offs*. The accuracy requirement is not baked in the definition of DP;

instead, it is a constraint made explicit for a specific task at hand when designing a differentially private data analysis.

An important property of DP is *composability*. Multiple differentially private data analyses can be composed with a graceful degradation of the privacy parameters (ϵ and δ). This property allows reasoning about privacy as a *budget*: a data analyst can decide how much privacy budget (the ϵ parameter) assigns to each of her analyses. The compositionality aspects of DP motivated the design of several programming frameworks [28, 52, 33, 24, 22, 7, 6, 44, 43, 31, 45, 65] and tools [39, 45, 42, 24] to help analysts design their own differentially private consults. At a high level, most of these programming frameworks and tools are based on a similar idea for reasoning about privacy: provide primitives for fundamental differentially private analyses as building blocks, and use composition properties to combine these building blocks. During composition, these systems ensure that the privacy cost of each data analysis sums up and that the total cost does not exceed the privacy budget. The programming frameworks also provide general support to further combine, through programming techniques, the different building blocks and the results of several data analyses. Differently, DP tools are optimized for specific tasks at the price of restricting the kinds of data analyses they can support.

Unfortunately, reasoning about accuracy is *less compositional* than reasoning about privacy. It depends both on the specific task at hand and on the specific accuracy measure that one is interested in offering to data analysts. Despite this, when restricted to specific mechanisms and specific forms of data analyses, one can measure accuracy through estimates given as *confidence intervals*, or error bounds. As an example, most of the standard mechanisms from the differential privacy literature come with theoretical confidence intervals or error bounds that can be exposed to data analysts to allow them to make informed decisions about the consults they want to run. This approach has been integrated in tools such as GUPT [45], PSI [24], and APEX [25]. Users of these tools, can specify the target confidence interval they want to achieve, and the tools adjust the privacy parameters accordingly, when sufficient budget is available¹.

In contrast, all the programming frameworks proposed so far [28, 52, 33, 24, 22, 7, 6, 44, 43, 31, 45, 65] do not offer any support to programmers or data analysts for tracking, and reasoning about, the accuracy of their data analyses. This phenomenon is in large part due to the complex nature of accuracy reasoning, concerning privacy analyses, when designing arbitrary data analyses that users of these frameworks may want to implement and execute. In this work, we address this limitation by building a programming framework for designing differentially private analyses, which supports a compositional form of reasoning about accuracy.

Contribution

Our main contribution is showing how programming frameworks can internalize the use of *probabilistic bounds* [15] for composing different confidence intervals or error bounds, in an automated way. Probabilistic bounds are part of the standard

¹APEX goes beyond this by also helping users select the right differentially private mechanism to achieve the required accuracy.

toolbox for the analysis of randomized algorithms. Specifically, they are the tools that differential privacy algorithms designers usually employ for the accuracy analysis of classical mechanisms [19, 19]. Two important probabilistic bounds are the *union bound*, which can be used to compose errors with no assumption on the way the random noise is generated, and *Chernoff bound*, which applies to the sum of random noise when the different random variables characterizing noise generation are statistically independent (see Section A.5). When applicable, and when the number of random variables grows, Chernoff bound usually gives a much “tighter” error estimation than the union bound.

Barthe *et al.* [8] have shown how the union bound can be internalized in a Hoare-style logic for reasoning about probabilistic imperative programs, and how this logic can be used to reason in a mechanized way about the accuracy of probabilistic programs, in particular, programs implementing differentially private primitives.

Building on this idea, we propose a programming framework where this kind of reasoning is automated, and can be combined with reasoning about privacy. Such a framework aims to offer programmers the tools they need to implement differentially private data analyses and explore their privacy-accuracy trade-offs, in a *compositional way*. This framework supports not only the use of union bound as a reasoning principle, but also the Chernoff bound when applicable. The insight is that probabilistic bounds relying on probabilistic independence of random variables can be smoothly integrated in a programming framework by using techniques from information-flow control [55] (in the form of taint analysis [56]). While these probabilistic bounds are not enough to express every accuracy guarantee one wants to formulate for arbitrary data analyses, they enable the inspection of a large class of user-designed programs. Our approach allows programmers to exploit the compositional nature of both privacy and accuracy, complementing in this way the support provided by tools such as GUPT [45], PSI [24], which yield confidence intervals estimate only at the level of individual queries; and by APEX [25], which issues confidence intervals estimate only at the level of a query workload for queries of the same type.

The described tool is materialized as a programming framework called DPella — an acronym for Differential Privacy in Haskell with accuracy— where data analysts can explore the privacy-accuracy trade-off while writing their differentially private data analyses. DPella provides several basic differentially private building blocks and composition techniques, which can be used by a programmer to design complex differentially private data analyses. The analyses that can be expressed in DPella are *data-independent* and can be built using primitives for counting, average, max, and any aggregation of their results.

DPella supports both pure-DP, with parameter ϵ , and approximate-DP, with parameters ϵ and δ . Accordingly, it supports the addition of both Laplace and Gaussian random noise, and the use of sequential or advanced [19] composition, respectively, together with parallel composition for both notions. For clarity, we will mainly focus on ϵ -DP and the Laplace mechanism, however other variants will be briefly discussed (see Section A.5.3). DPella is implemented as a library in the general-purpose language Haskell, a programming language that is well-known to support information-flow analyses [36, 54] easily. Furthermore, DPella is designed to be *extensible* by adding new primitives implementing advanced DP routines (see Section A.9).

To reason about privacy and accuracy, DPella provides two primitives responsible for interpreting programs (which implement data analyses) symbolically. DPella’s symbolic interpretation for privacy consists of decreasing the privacy budget of a query by deducing the required budget of its sub-parts. On the other hand, the accuracy interpretation uses as abstraction the *inverse Cumulative Distribution Function* (iCDF) representing an upper bound on the (theoretical) error that the program incurs when guaranteeing DP. A query’s iCDF is built out from the iCDFs of its components by using the *union bound* as the elemental composition principle. These interpretations provide overestimates of the corresponding quantities that they track. To make these estimates as precise as possible, DPella uses *taint analysis* to track the injection of noise and identify which variables are *statistically independent*. This information is used by DPella to replace *soundly*, when needed, the union bound with the *Chernoff bound*, something that to the best of our knowledge other program logics [8] or program analyses [57] also focusing on accuracy do not consider. We envision DPella’s accuracy estimations to be used in scenarios that align with those considered by tools like GUPT, PSI, and APEX.

In summary, our contributions are:

- Present DPella, a programming framework that allows data analysts to reason compositionally about privacy-accuracy trade-offs.
- Show how to use taint analysis to detect statistical independence of the noise that different primitives add, and how to use this information to achieve better error estimates.
- Inspect DPella’s expressiveness and error estimations by implementing PINQ-like queries from previous work [40, 28, 6] and workloads from the matrix mechanism [35, 30, 62].

To present DPella and its components, this document is structured as follows. Section A.2 provides a brief background on the notions of privacy and accuracy DPella considers. Section A.3 introduces DPella by showcasing its main features through simple examples. Section A.4 presents each of DPella’s primitives for the construction and execution of queries. Section A.5 explains how do we calculate accuracy concentration bounds and the accuracy-aware primitives that can be used by the data analysts. On Section A.6 we implement case studies from the literature revealing DPella’s advantages and limitations. Section A.7 introduces a new primitive that allows data analysts to test DPella’s accuracy estimations. Section A.8 shows DPella’s generalized API that allows data analyst to combine noisy values generated with different mechanisms. Following, on Section A.9 we discuss DPella’s limitations in detail together with possible extensions to the framework. Lastly, Section A.10 puts DPella in context while contrasting it with other approaches and frameworks.

Highlights This work builds on our previous paper “A Programming Framework for Differential Privacy with Accuracy Concentration Bounds” [38] that we have improved in its presentation and complemented with novel contributions summarized as follows:

- Comprehensive description of DPella’s components;
- Introduction of a new feature to tests DPella’s accuracy estimations (Section A.7), this way analysts will be able to measure the *tightness* of DPella’s bound;
- API updates including new accuracy combinators (Section A.5.1) giving more options to manipulate and modify noisy values without losing information of their accuracy;
- Description of (ϵ, δ) -DP and Gauss Mechanism integration (Section A.5.3) which showcases DPella’s flexibility to host other notions of differential privacy and mechanisms;
- Presentation of DPella’s generalized API (Section A.8) that facilitates the implementation of query plans involving results from different mechanisms.

A.2 Background

Differential privacy [18] is a quantitative notion of privacy that bounds how much a single individual’s private data can affect the result of a data analysis. More formally, we can define differential privacy as a property of a randomized query $\tilde{Q}(\cdot)$ representing the data analysis, as follow.

Definition A.1 (Differential Privacy (DP) [18]). *A randomized query $\tilde{Q}(\cdot) : \text{db} \rightarrow \mathbb{R}$ satisfies ϵ -differential privacy if and only if for any two datasets D_1 and D_2 in db , which differ in one row, and for every output set $S \subseteq \mathbb{R}$ we have*

$$\Pr[\tilde{Q}(D_1) \in S] \leq e^\epsilon \Pr[\tilde{Q}(D_2) \in S] \quad (\text{A.1})$$

In the definition above, the parameter ϵ determines a bound on the distance between the distributions induced by $\tilde{Q}(\cdot)$ when adding or removing an individual from the dataset—the farther away they are, the more at risk the privacy of an individual is, and vice versa. In other words, ϵ imposes a limit on the *privacy loss* that an individual can incur in, as a result of running a data analysis.

A standard way to achieve ϵ -differential privacy is adding some carefully calibrated noise to the result of a query. To protect all the different ways in which an individual’s data can affect the result of a query, the noise needs to be calibrated to the maximal change that the result of the query can have when changing an individual’s data. This is formalized through the notion of *sensitivity*.

Definition A.2 (Sensitivity [18]). *The (global) sensitivity of a query $Q(\cdot) : \text{db} \rightarrow \mathbb{R}$ is the quantity $\Delta_Q = \max\{|Q(D_1) - Q(D_2)| \text{ for } D_1, D_2 \text{ differing in one row}$*

The sensitivity gives a measure of the amount of noise needed to protect one individual’s data. Besides, in order to achieve differential privacy, it is also important the choice of the kind of noise that one adds. A standard approach is based on the addition of noise sampled from the Laplace distribution.

Theorem A.1 (Laplace Mechanism [18]). *Let $Q(\cdot) : \text{db} \rightarrow \mathbb{R}$ be a deterministic query with sensitivity Δ_Q . Let $\tilde{Q}(\cdot) : \text{db} \rightarrow \mathbb{R}$ be a randomized query defined as $\tilde{Q}(D) = Q(D) + \eta$, where η is sample from the Laplace distribution with mean $\mu = 0$ and scale $b = \Delta_Q/\epsilon$. Then \tilde{Q} is ϵ -differentially private.*

Notice that in the theorem above, for a given query, the smaller the ϵ is, the more noise $\tilde{Q}(\cdot)$ needs to inject in order to hide the contribution of one individual's data to the result—this protects privacy but degrades how meaningful the result of the query is—and vice versa. In general, the notion of *accuracy* can be defined more formally as follows.

Definition A.3 (Accuracy, see e.g. [19]). *Given an ϵ -differentially private query $\tilde{Q}(\cdot)$, a target query $Q(\cdot)$, a distance function $d(\cdot)$, a bound α , and the probability β , we say that $\tilde{Q}(\cdot)$ is $(d(\cdot), \alpha, \beta)$ -accurate with respect to $Q(\cdot)$ if and only if for all dataset D :*

$$\Pr[d(\tilde{Q}(D) - Q(D)) > \alpha] \leq \beta \quad (\text{A.2})$$

This definition allows one to express data independent error statements such as: with probability at least $1 - \beta$ the query $\tilde{Q}(D)$ diverge from $Q(D)$, in terms of the distance $d(\cdot)$, for less than α . Then, we will refer to α as the *error* and $1 - \beta$ as the *confidence probability* or simply *confidence*. In general, the lower the β is, i.e., the higher the confidence probability is, the higher the error α is.

As previously discussed, an important property of differential privacy is com-
poseability.

Theorem A.2 (Sequential Composition [18]). *Let $\tilde{Q}_1(\cdot)$ and $\tilde{Q}_2(\cdot)$ be two queries which are ϵ_1 - and ϵ_2 -differentially private, respectively. Then, their sequential composition $\tilde{Q}(\cdot) = (\tilde{Q}_1(\cdot), \tilde{Q}_2(\cdot))$ is $(\epsilon_1 + \epsilon_2)$ -differentially private.*

Theorem A.3 (Parallel Composition [28]). *Let $\tilde{Q}(\cdot)$ be a ϵ -differentially private query and $\text{data}_1, \text{data}_2$ be a partition of the set of data. Then, the query $\tilde{Q}_1(D) = (\tilde{Q}(D \cap \text{data}_1), \tilde{Q}(D \cap \text{data}_2))$ is ϵ -differentially private.*

Thanks to the composition properties of differential privacy, we can think about ϵ as a *privacy budget* that one can spend on a given data before compromising the privacy of individuals' contributions to that data. The *global* ϵ for a given program can be seen as the privacy budget for the entire data. This budget can be consumed by selecting the *local* ϵ to “spend” in each intermediate query. Thanks to the composition properties, by tracking the local ϵ that are consumed, one can guarantee that a data analysis will not consume more than the allocated privacy budget.

Given an ϵ , DPella gives data analysts the possibility to explore how to spend it on different queries and analyze the impact on accuracy. For instance, data analysts might decide to spend “more” epsilon on sub-queries which results are required to be more accurate, while spending “less” on the others. The next examples (inspired by the use of DP in network trace analyses [40]) show how DPella helps to quantify what “more” and “less” means.

A.3 DPella by example

DPella’s model considers two kinds of actors: *data curators*, owners of the private dataset that decide the global privacy budget and split it among the *data analysts*, the ones who write queries to mine useful information from the data and spend the budget they received. Analysts are not allowed to directly query the dataset, instead, they need to implement their analyses and send them to the curator who will execute them and give the results back.

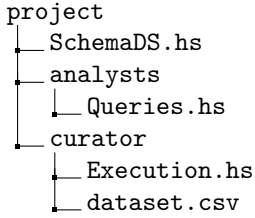


Figure A.1: File structure

From an implementation standpoint, it means that the analyses and their run functions are provided in different files, with different privileges. More specifically, Figure A.1 depicts a common file structure for the usage of DPella. File `SchemaDS.hs` contains the schema of the dataset owned by the curator, it does not contain private data, only the names of the tables and their respective attributes as a Haskell record type. For example, a dataset containing just one table called `Ages` with two attributes `name` (a `String` value) and `age` (an `Int`

value), will be encoded in `SchemaDS.hs` as follows, where `::` is used to describe the type of a term in Haskell:

```
data Ages = AgeRow { name :: String, age :: Int }
```

Since the structure of the dataset is not considered sensitive information, the file `SchemaDS.hs` can be accessed by both, the data owner and data analysts.

File `Queries.hs` contains the analyses that have been implemented by the data analysts, all of these queries should be parameterized by the dataset in which they will be later executed. Analysts will only have access to their implementations and the database schema. Lastly, file `Execution.hs` implements the run functions for the analyses at `Queries.hs`, this file is owned by the curator and has access to all other files in the directory, in particular, it has access to the real data—stored in `dataset.csv`.

A.3.1 Basic aggregations

For the following examples, we consider a dataset representing a *tcpdump* trace of packets where each row contains the information indicated by its schema:

```
data Tcpdump =
  TCPRow { id      :: Integer   , timestamp :: Double
          , src     :: IP       , dest      :: IP
          , protocol :: Integer  , size      :: Integer
          , payload  :: ByteString }
```

Counting

An analyst wanting to know the number of packets sent to *WikiLeaks*, with IP address `195.35.109.53`, can do so by writing a simple eps-differentially private query as follows:

```
import SchemaDS
import AnalystLP

wikileaks ::  $\epsilon \rightarrow$  Data 1 Tcpdump  $\rightarrow$  Query (Value Double)
wikileaks eps dataset = do
  byIP  $\leftarrow$  dpWhere (( $\equiv$  195.35.109.53)  $\circ$  dest) dataset
  dpCount eps byIP
```

First, we import file `SchemaDS` where `Tcpdump`'s description (previously presented) is stored. Then, we import `DPella`'s interface for analysts called `AnalystLP`, where `LP` indicates that we will use the Laplacian mechanism. Subsequently, we implement query `wikileaks` which takes as input the amount of privacy budget `eps` (of type ϵ) to be spent by the query and the dataset (of type `Data 1 Tcpdump`) where it will be computed; when executed, this query will yield results of type `Query (Value Double)`, that is, `DPella` computations of type `Double`—a more detailed explanation of `DPella`'s types could be found in the following sections. In query `wikileaks`, we use the primitive transformation² `dpWhere` to filter all rows whose `dest` attribute has a value equal to `195.35.109.53`, this operation returns a transformed dataset that we have called `byIP`. We proceed to perform the noisy count using primitive `dpCount` over the filtered dataset `byIP` while spending `eps` amount of privacy budget. The value of `eps` will—internally—determine the magnitude of noise to be added to the real count.

Having this general implementation, an analyst can write specific queries fixing the value of `eps`, for instance:

```
analysis1 = wikileaks 0.5
analysis2 = wikileaks 1
analysis3 = wikileaks 5
```

To execute these analyses the *data owner* needs to implement a function that loads the required dataset and execute analysts' queries, such a function will look like:

```
import SchemaDS
import CuratorLP (loadDS, dpEval)
import Queries

runAnalysis :: (Data 1 Tcpdump  $\rightarrow$  Query (Value Double))  $\rightarrow$   $\epsilon$ 
               $\rightarrow$  IO Double
```

²Anticipating on Section A.4, in our code we will usually use the red color for transformations, the blue color for aggregate operations, and the green color for combinators for privacy and accuracy.

```
runAnalysis query bud = do
  ds ← loadDS "hotspot.csv"
  dpEval query ds bud
```

Function `runAnalysis` takes as inputs the function to be executed, called `query`, and the global privacy budget `bud`; returning the randomized count as an `IO Double`. This function calls an auxiliary function `loadDS` (provided by DPella's interface for curators) to read file `hotspot.csv` which is then saved as a DPella's dataset in variable `ds`. Next, it uses DPella's primitive `dpEval` indicating which analysis will be performed, over which dataset, and what's the tolerance for the privacy loss.

Let's assume `hotspot.csv` have the information of 10,000 packets and 7 of them were directed to WikiLeaks' IP address. Then, when the data owner executes the analysis she would get results such as:

```
>runAnalysis analysis1 20
Value = 15.3
>runAnalysis analysis2 20
Value = 4.8
>runAnalysis analysis3 20
Value = 6.7
```

Which clearly exemplifies the effects of the selection of `eps` on the queries' results. Intuitively, the greater the `eps`, the closer we are to the real count of packets.

Sums

Suppose we are now interested in computing the amount of transmitted data. This is, we want to sum up the value of `size` column which indicates the length of the packets in bytes.

In DPella, to compute a sum, we need to determine first the range of the values—our framework supports only integer numbers' ranges, e.g., $[1, 10]$, $[-5, 30]$, etc. This information is needed to automatically calculate the sensitivity of sum queries at compile time, i.e., if every value is in the range $[a, b]$, the sensitivity of their addition is $\max\{|a|, |b|\}$. We specify ranges in DPella via the primitive `range`.

$$\text{range} :: (\text{IsInt } a, \text{IsInt } b, \text{IsNat} \mid b - a \mid, a \leq b) \Rightarrow \text{Range } a \ b$$

This function receives no arguments since the range is indicated at the type-level with type constraints of the form `IsNat n` for strictly positive integer numbers, and `IsInt n` for positive and negative integers. Then, to create ranges we need to use type applications, e.g.,

```
range1 = range @(:+ 1) @(:+ 10)
range2 = range @(:- 5) @(:+ 30)
```

Here, functions `:+` and `:-` are used to specify the sign of the range's limits.

For the example of packets size, the data curator indicates that the range of size attribute goes from 40 to 35,000 bytes, then we define our query as follows:

```
totalBytes eps dataset = do
  dpSum eps (range @(:+: 40) @(:+: 35000)) size dataset
```

Function `totalBytes` uses primitive `dpSum` to compute the noisy sum of `size` attribute—whose values are ranging from 40 to 35000 bytes—over the indicated `dataset`. The way this query should be executed does not vary from the execution of the analyses derived from function `wikileaks`, thus is omitted.

Changing the question to focus on an specific protocol might require an adjustment on the range to be specified. For instance, if instead we want to inspect the total amount of data transmitted through Kerberos’ authentication protocol, which uses port 88, we should use the fact that this port transmits packets of at most of 1465 bytes. Hence, we will need to update our query accordingly

```
totalBytesKerberos eps dataset = do
  kerberos ← dpWhere ((≡ 88) ∘ protocol) dataset
  dpSum eps (range @(:+: 40) @(:+: 1465)) size kerberos
```

In function `totalBytesKerberos` we will first filter the dataset to obtain the information regarding port 88, then we perform the noisy sum over the filtered data. Observe that we are defining a query with less global sensitivity than the one implemented in function `totalBytes`, thus, if given the same `eps`, less noise will be added to the results of the analyses deriving from function `totalBytesKerberos`. Having a notion of the order of magnitude in which the result of a sum ranges becomes handy when reasoning about the accuracy of the query. In the following examples we depict how an analyst can use DPella to inspect the error of her queries, check out for miscalculations on the consumption of the privacy budget, and more.

A.3.2 Cumulative Distribution Function

Considering the same dataset `Tcpdump` we would like to inspect—in a differentially private manner—the packet’s length distribution by computing its Cumulative Distribution function (CDF), defined as $CDF(x) = \text{number of records with value} \leq x$. Hence, we are just interested in the values of the attribute `size`. McSherry and Mahajan [40] proposed three different ways to approximate (due to the injected noise) CDFs with DP, and they argued for their different levels of accuracy. For simplicity, we revise two of these approximations to show how DPella can assist in showing the accuracy of these analyses.

Sequential CDF

A simple approach to compute the CDF consists in splitting the range of lengths into bins and, for each bin, count the number of records that are \leq bin. A natural way to make this computation differentially private is to add independent Laplace noise to each count.

We show how to do this using DPella in Figure A.2a. We define a function `cdf1` which takes as input the list of bins describing size ranges, the amount of budget `eps` to be spent by the entire query, and the `dataset` where it will be computed. For

```

1 cdf1 bins eps dataset = do
2   sizes ← dpSelect size dataset
3   counts ← sequence [do elems ← dpWhere (≤ bin) sizes
4                      dpCount localEps elems | bin ← bins]
5   return (norm∞ counts)
6   where localEps = eps / length bins

```

(a) Sequential approach

```

7 cdf2 bins eps dataset = do
8   sizes ← dpSelect ((≤ max bins) ∘ size) dataset
9   -- parts :: Map Integer (Value Double)
10  parts ← dpPartRepeat (dpCount eps) bins assignBin sizes
11  let counts      = Map.elems parts
12      cumulCounts = [add (take i counts) | i ← [1..length counts]]
13  return (norm∞ cumulCounts)

```

(b) Parallel approach

Figure A.2: CDF's implementations

now, we assume that we have a fixed list of bins for packets' length. Function `cdf1` uses the primitive transformation `dpSelect` to obtain from the dataset the length of each packet via a selector function, in this case it is just the column of interest `size`. This computation results in a new dataset `sizes`. Then, we create a counting query for each bin using the primitive `dpWhere`. This filters all records that are less than the bin under consideration ($\leq \text{bin}$). Finally, we perform a noisy count using primitive `dpCount`. The noise injected by the primitive `dpCount` is calibrated so that the execution of `dpCount` is `localEps`-DP (line 6³). The function sequence then takes the list of queries and compute them sequentially collecting their results in a list—to create a list of noisy counts. We then return this list. The combinator `norm∞` in line 5 is used to mark where we want the accuracy information to be collected, but it does not have any impact on the actual result of the cdf.

To ensure that `cdf1` is `eps`-differential privacy, we distributed the given budget `eps` evenly among the sub-queries (this is done in lines 4 and 6). However, a data analyst may forget to do so, e.g., she can define `localEps = eps`, and in this case the final query is $(\text{length bins}) * \text{eps}$ -DP, which is a significant change in the query's privacy price. To prevent such budget miscalculations or unintended expenditure of privacy budget, DPella provides the analyst with the function `budget` (see Section A.4) that, given a query, statically computes an upper bound on how much budget it will spend. To see how to use this function, consider the function `cdf1` and a its modified version `cdf1'` with `localEps = eps`. Suppose that we want to compute how much budget will be consumed by running them on a list of 10 bins

³The casting operation `fromIntegral` is omitted for clarity

(identified as `bins10`) and a symbolic dataset `symDataset`. Then, the data analyst can ask this as follows:

```
>budget (cdf1 bins10 1 symDataset)
 $\epsilon = 1$ 
>budget (cdf1' bins10 1 symDataset)
 $\epsilon = 10$ 
```

The function `budget` will not execute the query, it simply performs an static analysis on the code of the query by symbolically interpreting it. The static analysis uses information encoded by the *type* of `symDataset` (explained in Section A.4), that, in this particular case, will be provided by `Tcpdump`'s schema.

DPella also provides primitives to statically explore the accuracy of a query. The function `accuracy` takes a noisy query $\tilde{Q}(\cdot)$ and a probability β and returns an estimate of the (theoretical) error that can be achieved with confidence probability $1 - \beta$. Suppose we want to estimate the error we will incur in by running `cdf1` with a budget of $\epsilon = 1$ with the same list of bins and symbolic dataset as before, and we want to have this estimate for $\beta = 0.05$ and $\beta = 0.2$, respectively. Then, the data analyst can ask this as follow:

```
>accuracy (cdf1 bins10 1 symDataset) 0.05
 $\alpha = 53$ 
>accuracy (cdf1 bins10 1 symDataset) 0.2
 $\alpha = 40$ 
```

Since the result of the query is a vector of counts, we measure the error α in terms of ℓ_∞ distance with respect to the CDF without noise. This is the max difference that we can have in a bin due to the noise. The way to read the information provided by DPella is that with confidence 95% and 80%, we have errors 53 and 40, respectively. These error bounds can be used by a data analyst to figure out the exact set of parameters that would be useful for her task.

Parallel CDF

Another way to compute a CDF is by first generating an histogram of the data according to the bins, and then building a cumulative sum for each bin. To make this function private, an approach could be to add noise at the different bins of the histogram, rather than to the cumulative sums themselves, so that we could use the parallel composition, rather than the sequential one [40], which we show how to implement in DPella in Figure A.2b—where double-dashes are used to introduce single-line comments.

In `cdf2`, we first select all the packages whose length is smaller than the maximum bin (line 8), then we partition the data accordingly to the given list of bins (line 10). To do this, we use `dpPartRepeat` operator to create as many (disjoint) datasets as given bins, where each record in each partition belongs to the range determined by an specific bin—where the record belongs is determined by the function `assignBin :: Integer \rightarrow Integer`. After creating all partitions, the primitive

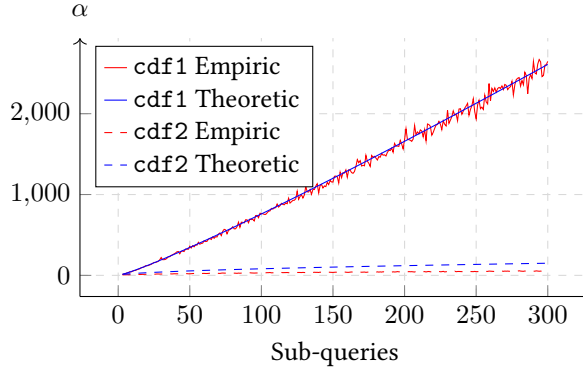


Figure A.3: Error comparison (95% confidence)

`dpPartRepeat` computes the given query `dpCount` eps in *each partition*—the name `dpPartRepeat` comes from repetitively calling `dpCount` eps as many times as partitions we have. As a result, `dpPartRepeat` returns a finite map where the keys are the bins and the elements are the noisy count of the records per partition—i.e., the histogram. In what follows (lines 12–13), we compute the cumulative sums of the noisy counts using the DPella primitive `add`, and finally we build and return the list of values denoting the CDF.

The privacy analysis of `cdf2` is similar to the one of `cdf1`. The accuracy analysis, however, is more interesting: first it gets error bounds for each cumulative sum, then these are used to give an error bound on the maximum error of the vector. For the error bounds on the cumulative sums DPella uses either the union bound or the Chernoff bound, depending on which one gives the lowest error. For the maximum error of the vector, DPella uses the union bound, similarly to what happens in `cdf1`. A data analyst can explore the accuracy of `cdf2`.

```
>accuracy (cdf2 bins10 1 symDataset) 0.05
alpha = 22
>accuracy (cdf2 bins10 1 symDataset) 0.2
alpha = 20
```

Exploring the privacy-accuracy trade-off

Let us assume that a data analyst is interested in running a CDF with an error bounded with 90% confidence, i.e., with $\beta = 0.1$, having three bins (named `bins3`), and $\epsilon = 1$. With those assumptions in mind, which implementation should she use? To answer that question, the data analyst can ask DPella:

```
>accuracy (cdf1 bins3 1 symDataset) 0.1
alpha = 11
>accuracy (cdf2 bins3 1 symDataset) 0.1
alpha = 12
```

So, the analyst would know that using `cdf1` in this case would give, likely, a lower error. Suppose further that the data analyst realize that she prefers to have a finer granularity and have 10 bins, instead of only 3. Which implementation should she use? Again, she can compute:

```
>accuracy (cdf1 bins10 1 symDataset) 0.1
 $\alpha = 46$ 
>accuracy (cdf2 bins10 1 symDataset) 0.1
 $\alpha = 20$ 
```

So, the data analyst would know that using `cdf2` in this case would give, likely, a lower error. One can also use `DPella` to show a comparison between `cdf1` and `cdf2` in terms of error when we keep the privacy parameter fixed and we change the number of bins, where `cdf2` gives a better error when the number of bins is large [40] as illustrated in Figure A.3. In the figure, we also show the empirical error to confirm that our estimate is tight—the oscillations on the empirical `cdf1` are given by the relative small (300) number of experimental runs we consider.

Now, what if the data analyst chooses to use `cdf2` because of what we discussed before but she realizes that she can afford an error $\alpha \leq 50$; what would be then the epsilon that gives such α ? One of the feature of `DPella` is that the analyst *can write a simple program that finds it by repetitively calling `accuracy` with different epsilons*—this is one of the advantages of providing a programming framework. These different use cases shows the flexibility of `DPella` for different tasks in private data analyses.

A.4 Privacy

`DPella` is designed to help data analysts to have an informed decision about how to spend their budget based on exploring the trade-offs between privacy and accuracy. In this section, we introduce `DPella`'s primitives and design principles responsible to ensure differential privacy of queries written by data analysts.

A.4.1 Components of the API

Figure A.4 shows part of `DPella` API. `DPella` introduces two abstract data types to respectively denote datasets and queries:

```
data Data s r -- datasets
data Query a -- queries
```

The attentive reader might have observed that the API also introduces the data type `Value a`. This type is used to capture values resulting from data aggregations. However, we defer its explanation for Section A.5 since it is only used for accuracy calculations—for this section, readers can consider the type `Value a` as isomorphic to the type `a`. It is also worth noticing that the API enforces an invariant by construction: *it is not possible to branch on results produced by aggregations*—observe

```

-- Transformations (data analyst)
dpWhere    :: (r → Bool) → Data s r → Query (Data s r)
dpGroupBy  :: Eq k ⇒ (r → k) → Data s r → Query (Data (2*s) (k, [r]))
dpIntersect :: Eq r ⇒ Data s1 r → Data s2 r → Query (Data (s1+s2) r)
dpSelect   :: (r → r') → Data s r → Query (Data s r')
dpUnion    :: Data s1 r → Data s2 r → Query (Data (s1+s2) r)
dpPart     :: Ord k ⇒ (r → k) → Data s r → Map k (Data s r)
           → Query (Value a) → Query (Map k (Value a))

-- Aggregations (data analyst)
dpCount    :: Stb s ⇒ ε → Data s r → Query (Value Double)
dpSum      :: Stb s ⇒ ε → Range a b → (r → Double) → Data s r
           → Query (Value Double)
dpAvg      :: Stb s ⇒ ε → Range a b → (r → Double) → Data s r
           → Query (Value Double)
dpMax      :: Eq a ⇒ ε → Responses a → (r → a) → Data 1 r
           → Query (Value a)

-- Budget
budget     :: Query a → ε

-- Execution (data curator)
dpEval    :: (Data 1 r → Query (Value a)) → [r] → ε → IO a

```

Figure A.4: DPella API: Part I

that there is no primitive capable to destruct a value of type `Value a`. While it might seem restrictive, it enables to write counting queries, which are the bread and butter of statistical analysis and have been the focus of the majority of the work in DP. Section A.9 discusses, however, how to lift this limitation for specific analyses.

Values of type `Data s r` represent sensitive datasets with *accumulated stability* `s`, where each row is of type `r`. Accumulated stability, on the other hand, is instantiated to type-level positive natural numbers, i.e., `1`, `2`, etc. Stability is a measure that captures the number of rows in the dataset that could have been affected by transformations like selection or grouping of rows. In DP research, stability is associated with dataset transformations rather than with datasets themselves. In order to simplify type signatures, DPella uses the type parameter `s` in datasets to represent the accumulated stability of the transformations for which datasets have gone through—as done in [20]. Different than, e.g., PINQ [28], one novelty of DPella is that it computes stability *statically* using Haskell’s type-system.

Values of type `Query a` represent *computations*, or queries, that yield values of type `a`. Type `Query a` is a monad [44], and because of this, computations of type `Query a` are built by two fundamental operations:

```

return :: a → Query a
(>=>) :: Query a → (a → Query b) → Query b

```

The operation `return x` outputs a query that just produces the value `x` without causing side-effects, i.e., without touching any dataset. The function `(>=>)`—called *bind*—is used to sequence queries and their associated side-effects. Specifically,

`qp >>= f` executes the query `qp`, takes its *result*, and passes it to the function `f`, which then returns a second query to run. Some languages, like Haskell, provide syntactic sugar for monadic computations known as **do**-notation. For instance, the program `qp1 >>= (λx1 → qp2 >>= (λx2 → return (x1, x2)))`, which performs queries `qp1` and `qp2` and returns their results in a pair, can be written as `do x1 ← qp1; x2 ← qp2; return (x1, x2)` which gives a more “imperative” feeling to programs. We split the API in four parts: transformations, aggregations, budget prediction, and execution of queries—see next section for the description of API’s accuracy components. The first three parts are intended to be used by data analysts, while the last one is intended to be *only* used by data curators⁴.

A.4.2 Transformations

The primitive `dpWhere` filters rows in datasets based on a predicate functions (`r → Bool`). The created query (of type `Query (Data s r)`) produces a dataset with the same row type `r` and accumulated stability `s` as the dataset given as argument (`Data s r`). Observe that if we consider two datasets which differ in `s` rows in two given executions, and we apply `dpWhere` to both of them, we will obtain datasets that will still differ in at most `s` rows—thus, the accumulated stability remains the same. The primitive `dpGroupBy` returns a dataset where rows with the same key are grouped together. The functional argument (of type `r → k`) maps rows to keys of type `k`. The rows in the return dataset (`Data (2*s) (k, [r])`) consist of key-rows pairs of type `(k, [r])`—syntax `[r]` denotes the type of lists of elements of type `r`. What appears on the left-hand side of the symbol `⇒` are type constraints. They can be seen as static demands for the types appearing on the right-hand side of `⇒`. Type constraint `Eq k` demands type `k`, denoting keys, to support equality; otherwise grouping rows with the same keys is not possible. The accumulated stability of the new dataset is multiplied by `2` in accordance with stability calculations for transformations [28, 20]—observe that `2*s` is a type-level multiplication done by a type-level function (or type family [21]) `*`, in other words, it is an arithmetic operation computed at compile time. Our API also considers transformations similar to those found in SQL like intersection (`dpIntersect`), union (`dpUnion`), and selection (`dpSelect`) of datasets, where the accumulated stability is updated accordingly. Providing a general join transformation is known to be challenging [28, 46, 10, 32]. The output of a join may contain duplicates of sensitive rows, which makes difficult to bound the accumulated stability of datasets. However, and similar to PINQ, DPella supports a limited form of joins, where a limit gets imposed on the number of output records mapped under each key in order to obtain stability. For brevity, we skip its presentation and assume that all the considered information is contained by the rows of given datasets.

A.4.3 Partition

Primitive `dpPart` deserves special attention. This primitive is a mixture of a transformation and aggregations since it partitions the data (transformation) to subse-

⁴A separation that can be enforced via Haskell modules [36]

```

1 q ::  $\epsilon \rightarrow [\text{Color}] \rightarrow \text{Data s r Double} \rightarrow \text{Query (Map Color Double)}$ 
2 q eps bins dataset = dpPart id dataset dps
3   where dps = fromList [(c,  $\lambda ds \rightarrow \text{dpCount eps dataset}$ ) | c  $\leftarrow$  bins]
4       -- dps = fromList [(c,  $\lambda ds \rightarrow \text{dpCount eps ds}$ ) | c  $\leftarrow$  bins]

```

Figure A.5: DP-histograms by using `dpPart`

quently apply aggregations on each of them. More specifically, it splits the given dataset (`Data s r`) based on a row-to-key mapping ($r \rightarrow k$). Then, it takes each partition for a given key k and applies it to the corresponding function `Data s r \rightarrow Query (Value a)`, which is given as an element of a key-query mapping of type `Map k ((Data s r) \rightarrow Query (Value a))`. Subsequently, it returns the values produced at every partition as a key-value mapping (`Query (Map k (Value a))`). The primitive `dpPartRepeat`, used by the examples in Section A.3, is implemented as a special case of `dpPart` and thus we do not discuss it further.

Partition is one of the most important operators to save privacy budget. It allows to run the same query on a dataset's partitions but only paying for one of them—recall Theorem A.3. The essential assumption that makes this possible is that every query runs on *disjoint* datasets. Unfortunately, data analysts could ignore this assumption when writing queries.

To illustrate this point, we present the code in Figure A.5. Query q produces an ϵ -DP histogram of the colors found in the argument dataset, which rows are of type `Color` and variable `bins` enumerates all the possible values of such type. The code partitions the dataset by using the function `id :: Color \rightarrow Color` (line 2) and executes the aggregation counting query (`dpCount`) in each partition (line 3)—function `fromList` creates a map from a list of pairs. The attentive reader could notice that `dpCount` is applied to the original dataset rather than the partitions. This type of errors could lead to break privacy as well as inconsistencies when estimating the required privacy budget. A correct implementation consists on executing `dpCount` on the corresponding partition as shown in the commented line 4.

To catch coding errors as the one shown above, DPella deploys an static analysis of information-flow control (IFC) similar to that provided by MAC [53]. IFC ensures that queries run by `dpPart` do not perform queries on shared datasets by attaching provenance labels to datasets `Data s r` indicating to which part of the query they are associated with and propagates that information accordingly.

Coming back to our previous example (see Figure A.5), the IFC analysis will assign the provenance of dataset in q to the top-level fragment of the query rather than to sub-queries executed in each partition—and DPella will raise an error at compile time when `ds` is accessed by the sub-queries! Instead, if we comment line 3 and uncomment line 4, the query q will be successfully run by DPella (when there is enough privacy budget) since every partition is only accessing their own partitioned data (denoted by variable `ds`).

The implemented IFC mechanism is *transparent* to data analysts and curators, i.e., they do not need to understand how it works. Analysts and curators only need

to know that, when the IFC analysis raises an alarm, is due to a possibly access to non-disjoint datasets when using `dpPart`.

A.4.4 Aggregations

DPella presents primitives to count (`dpCount`), sum (`dpSum`), and average (`dpAvg`) rows in datasets. These primitives take an argument `eps :: ϵ` , a dataset, and build a Laplace mechanism which is `eps`-differentially private from which a noisy result gets returned as a term of type `Value Double`. The purpose of data type `Value a` is two fold: to encapsulate noisy values of type `a` originating from aggregations of data, and to store information about its accuracy—intuitively, how “noisy” the value is (explained in Section A.5). The injected noise of these queries gets adjusted depending on three parameters: the value of type `ϵ` , the accumulated stability of the dataset `s`, and the sensitivity of the query (recall Definition A.2). More specifically, the Laplace mechanism used by DPella uses accumulated stability `s` to scale the noise, i.e., it consider `b` from Theorem A.1 as $b = s \cdot \frac{\Delta_Q}{\epsilon}$. The sensitivity of DPella’s aggregations are either hard-coded into the implementation—similar to what PINQ does—or calculated statically. The sensitivities of `dpSum` and `dpAvg` are determined by the range of the values under consideration i.e., for the indicated Range `a b`, the sensitivity is computed as $\max \{ |a|, |b| \}$ and $|b-a|$, respectively. This is enforced by applying a clipping function (`r → Double`). This function ensures that the values under scrutiny fall into the interval $[a, b]$ before (and, for `dpAvg`, after) executing the query. The sensitivity of `dpCount` and `dpMax` is set to `1`. To implement the Laplace mechanism, the type constrain `Stb s` in `dpCount`, `dpSum`, and `dpAvg` demands the accumulated stability parameter `s` to be a type-level natural number in order to obtain a term-level representation when injecting noise. Finally, primitive `dpMax` implements report-noisy-max [19]. This query takes a list of possible responses (`Responses a` is a type synonym for $[a]$) and a function of type `r → a` to be applied to every row. The implementation of `dpMax` adds uniform noise to every score—in this case, the amount of rows *voting* for a response—and returns the *response* with the highest noisy score. This primitive becomes relevant to obtain the winner option in elections without singling out any voter. However, it requires that the accumulated stability of the dataset to be `1` in order to be sound [8]. DPella guarantees such requirement by typing: the type of the given dataset as argument is `Data 1 r`.

A.4.5 Privacy budget and execution of queries

The primitive `budget` statically computes how much privacy budget is required to run a query. It is worth notice that DPella returns an upper bound of the required privacy budget rather than the exact one—an expected consequence of using a type-system to compute it and provide early feedback to data analysts. Finally, the primitive `dpEval` is used by data curators to run queries (`Query a`) under given privacy budgets (`ϵ`), where datasets are just lists of rows ($[r]$). It assumes that the initial accumulated stability as `1` (`Data 1 r`) since the dataset has not yet gone through any transformation, and DPella will automatically calculate the accumulated stability for datasets affected by subsequent transformations via the Haskell’s type system.

This primitive returns a computation of type `IO a`, which in Haskell are computations responsible to perform side-effects—in this case, obtaining randomness from the system in order to implement the Laplace mechanism.

A.4.6 Implementation

DPella is implemented as a *deep embedded domain-specific language* (EDSL) in Haskell. Due to such design choice, data analysts can piggyback on Haskell’s infrastructure to build queries in a creative way. For instance, it is possible to leverage on any of Haskell’s pure functions. The following one-liner (of type `Query [Value Double]`) uses several Haskell functions to filter a dataset `ds` in several (possibly non-disjoint) ways according to a list of predicates `ps :: [r → Bool]`, then for each filtered version of `ds` it performs a noisy count spending `eps` on each count.

```
mapM (flip dpSelect ds >=> dpCount eps) ps
```

The high-order functions `flip`, `mapM`, and `(>=>)` are standard in Haskell and represent a function who switches arguments, the monadic versions of `map`, and the Kleisli arrow, respectively. Despite DPella being a first-order interface, data analysts can use Haskell’s high-order functions to compactly describe queries.

A.5 Accuracy

DPella uses the data type `Value a` responsible to store a result of type `a` as well as information about its accuracy. For instance, a term of type `Value Double` stores a noisy number (e.g., coming from executing `dpCount`) together with its accuracy in terms of *a bound on the noise introduced to protect privacy*.

DPella provides a static analysis capable to compute the accuracy of queries via the following function

```
accuracy :: Query (Value a) → β → α
```

which takes as an argument a query and returns a function, called *inverse Cumulative Distribution Function* (iCDF), capturing the theoretical error α for a given confidence $1-\beta$. Function `accuracy` does not execute queries but rather symbolically interpret all of its components in order to compute the accuracy of the result based on the sub-queries and how data gets aggregated. DPella follows the principle of improving accuracy calculations by detecting statistical independence. For that, it implements taint analysis [56] in order to track if values were drawn from statistically independent distributions. DPella’s primitives involving accuracy calculations are presented in Figure A.6 and will be described in the following subsections.

A.5.1 Accuracy calculations

DPella starts by generating iCDFs at the time of running aggregations based on the following known result of the Laplace mechanism:

```

-- Accuracy analysis (data analyst)
accuracy :: Query (Value a) → β → α

-- Norms (data analyst)
norm∞ :: [Value Double] → Value [Double]
norm2  :: [Value Double] → Value [Double]
norm1  :: [Value Double] → Value [Double]
rmsd   :: [Value Double] → Value [Double]

-- Accuracy combinators (data analyst)
add    :: [Value Double] → Value Double
sub    :: [Value Double] → Value Double
neg    :: Value Double  → Value Double
scalar :: Value Double → Double → Value Double

```

Figure A.6: DPella API: Part II

Definition A.4 (Accuracy for the Laplace mechanism). *Given a randomized query $\tilde{Q}(\cdot) : \text{db} \rightarrow \mathbb{R}$ implemented with the Laplace mechanism as in Theorem A.1, we have that*

$$\Pr \left[|\tilde{Q}(D) - Q(D)| > \log\left(\frac{1}{\beta}\right) \cdot \frac{\Delta_Q}{\epsilon} \right] \leq \beta \quad (\text{A.3})$$

Recall that the Laplace mechanism used by DPella utilizes accumulated stability s to scale the noise, i.e., it considers b from Theorem A.1 as $b = s \cdot \frac{\Delta_Q}{\epsilon}$. Consequently, DPella stores the iCDF $\lambda\beta \rightarrow \log\left(\frac{1}{\beta}\right) \cdot s \cdot \frac{\Delta_Q}{\epsilon}$ for the values of type `Value Double` returned by aggregation primitives like `dpCount`, `dpSum`, and `dpAvg`. However, queries are often more complex than just calling aggregation primitives—as shown by `cdf2` in Figure A.2b. In this light, DPella provides combinators responsible to aggregate noisy values, while computing its iCDFs based on the iCDFs of the arguments. Figure A.6 shows DPella API when dealing with accuracy.

Norms

DPella exposes primitives to aggregate the magnitudes of several errors predictions into a *single* measure—a useful tool when dealing with vectors. Primitives `norm∞`, `norm2`, and `norm1` take a list of values of type `Value Double`, where each of them carries accuracy information, and produces a *single value* (or vector) that contains a list of elements (`Value [Double]`) whose accuracy is set to be the well-known ℓ_∞ -, ℓ_2 -, ℓ_1 -norms, respectively. Finally, primitive `rmsd` implements *root-mean-square deviation* among the elements given as arguments. In our examples, we focus on using `norm∞`, but other norms are available for the taste, and preference, of data analysts.

Adding values

The primitive `add` aggregates values and, in order to compute the accuracy of the addition, it tries to apply the Chernoff bound if all the values are statistically inde-

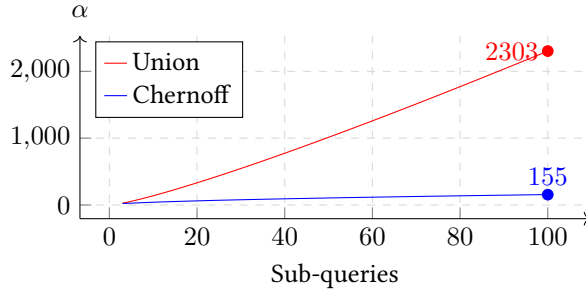


Figure A.7: Union vs. Chernoff bounds

pendent; otherwise, it applies the union bound. More precisely, for the next definitions we assume that primitive `add` receives n terms $v_1 :: \text{Value Double}$, $v_2 :: \text{Value Double}$, ..., $v_n :: \text{Value Double}$. Importantly, since we are calculating the theoretical error, we should consider random variables rather than specific numbers. The next definition specifies how `add` behaves when applying union bound.

Definition A.5 (`add` using union bound). *Given $n \geq 2$ random variables V_j with their respective $iCDF_j$, where $j \in 1 \dots n$, and $\alpha_j = iCDF_j(\frac{\beta}{n})$, then the addition $Z = \sum_{j=1}^n V_j$ has the following accuracy:*

$$\Pr[|Z| > \sum_{j=1}^n \alpha_j] \leq \beta \quad (\text{A.4})$$

Observe that to compute the $iCDF$ of Z , the formula uses the $iCDFs$ from the operands applied to $\frac{\beta}{n}$. Union bound makes no assumption about the distribution of the random variables V_j .

In contrast, the Chernoff bound *often* provides a tighter error estimation than the commonly used union bound when adding several *statistically independent* queries *sampled from a Laplace distribution*. To illustrate this point, Figure A.7 shows that difference for the `cdf2` function we presented in Section A.3 with $\epsilon = 0.5$ (for each DP sub-query) and $\beta = 0.1$. Clearly, the Chernoff bound is asymptotically much better when estimating accuracy, while the union bound works best with a reduced number of sub-queries—observe how lines get crossed in Figure A.7. In this light, and when possible, DPella computes both union bound and Chernoff bound and selects the tighter error estimation. However, to apply Chernoff bound, DPella needs to be certain that the events are independent. Before explaining how DPella detects that, we give a specification of the formula we use for Chernoff.

Definition A.6 (`add` using Chernoff bound [13]). *Given $n \geq 2$ independent random variables $V_j \sim \text{Lap}(0, b_j)$, where $j \in 1 \dots n$, $b_M = \max\{b_j\}_{j=1 \dots n}$, and $\nu > \max\left\{\sqrt{\sum_{j=1}^n b_j^2}, b_M \cdot \sqrt{\ln\left(\frac{2}{\beta}\right)}\right\}$, then the addition $Z = \sum_{j=1}^n V_j$ has the following accuracy:*

$$\Pr\left[|Z| > \nu \cdot \sqrt{8 \cdot \ln\left(\frac{2}{\beta}\right)}\right] \leq \beta \quad (\text{A.5})$$

DPella uses the value $\nu = \max \left\{ \sqrt{\sum_{j=1}^n b_j^2}, b_M \cdot \sqrt{\ln \left(\frac{2}{\beta} \right)} \right\} + 0.00001$ to satisfy the conditions of the definition above when applying the Chernoff bound—any other positive increment to the computed maximum works as well⁵. It is worth mentioning that DPella’s error estimations for the sums of noisy values rely on available concentration bounds. Hence, even though there exist better approximations for the error of adding random variables (e.g., dependency-dependent bounds for dependent variables [34]), currently, union and Chernoff bounds are the only statistical tools that can be used out of the box.

Lastly, to support subtraction, DPella provides primitive `neg` responsible to change the sign of a given value. And `sub` uses the results of `neg` and `add` to subtract a list of values. Following we explain how DPella checks that values come from statistically independent sampled variables.

Detecting statistical independence

To detect statistical independence, we apply taint analysis when considering terms of type `Value a`. Specifically, every time a result of type `Value Double` gets generated by an aggregation query in DPella’s API (i.e., `dpCount`, `dpSum`, etc.), it gets assigned a label indicating that it is *untainted* and thus statistically independent. The label also carries information about the scale of the Laplace distribution from which it was sampled—useful information when applying Definition A.6. When the primitive `add` receives all untainted values as arguments, the accuracy of the aggregation is determined by the best estimation provided by either the union bound (Definition A.5) or the Chernoff bound (Definition A.6). Importantly, values produced by `add` are considered *tainted* since they depend on other results. When `add` receives any tainted argument, it proceeds to estimate the error of the addition by just using union bound.

```

1 totalCount :: Query (Value Double)
2 totalCount = do
3   v1 ← dpCount 0.3 ds1
4   v2 ← dpCount 0.25 ds2
5   ...
6   v100 ← dpCount 0.5 ds100
7   return (add [v1, v2, ..., v100])

```

Figure A.8: Combination of sub-queries results

As an example, Figure A.8 presents the query plan `totalCount` which adds the results of hundred `dpCount` queries over different datasets, namely `ds1`, `ds2`, ..., `ds100`. (The ... denotes code intentionally left unspecified.) The code calls the primitive `add` with the results of calling `dpCount` — we use $[x_1, x_2, x_3]$ to denote the list with elements x_1 , x_2 , and x_3 . What would it be then the theoretical error of `totalCount`? The accuracy calculation depends on whether all the values

are untainted in line 7. When no dependencies are detected between v_1 , v_2 , ..., v_{100} , namely all the values are untainted, DPella applies Chernoff bound in order to give a tighter error estimation. Instead, for instance, if v_3 was computed as an augmentation of v_1 by a factor of 5, this is, let $v_3 = \text{scalar } v_1 \ 5$. Then, line 7 applies

⁵There are perhaps other ways to compute the Chernoff bound for the sum of independent Laplace distributions, changing this equation in DPella does not require major work.

union bound since v_3 is a tainted value—its noise is not freshly sampled but rather inherit from v_1 ’s noise. With taint analysis, DPella is capable to detect dependencies among terms of type `Value Double`, and leverages that information to apply different concentrations bounds. The next section formally defines such a procedure.

A.5.2 Implementation

The accuracy analysis consists on *symbolically* interpreting a given query, calculating the accuracy of individual parts, and then combining them using our taint analysis. We introduce two polymorphic symbolic values: $\mathcal{D} :: \text{Data } s \ r$ and $S[\text{iCDF}, s, \text{ts}] :: \text{Value } a$. Symbolic dataset \mathcal{D} represents concrete datasets arising from data transformations. A symbolic value $S[\text{iCDF}, s, \text{ts}]$ represents concrete values with tags ts and a iCDF which is computed assuming a noise scale s . Tags are used to detect the provenance of symbolic values and when they arise from different *noisy sources*.

Function `accuracy` takes queries producing results of type `Value a`. Such queries are essentially built by performing data aggregation queries (e.g., `dpCount`) preceded by a (possibly empty) sequence of other primitives like data transformations⁶. Figures A.9 and A.10 show the interesting parts of our analysis. Given a *well-typed* query $q :: \text{Query } (\text{Value } a)$, `accuracy` $q = \text{iCDF}$ where $q \triangleright S[\text{iCDF}, s, \text{ts}]$ for some s and ts . The rules in A.9 are mainly split into two cases: considering data aggregation queries and sequences of primitives glued together with $(\gg=)$.

The symbolic interpretation of `dpCount` is captured by rule `DPCOUNT`—see Figure A.9a. This rule populates the iCDF of the return symbolic value with the corresponding error calculations for Laplace as presented in Definition A.4 (with the scale adjusted with the accumulated stability). Observe that it extracts the stability information from the type of the considered dataset ($\text{ds} :: \text{Data } s \ r$) and attaches a fresh tag indicating an independently generated noisy value. The symbolic interpretation of `dpSum` and `dpAvg` proceeds similarly to `dpCount` and we thus omit them for brevity.

Rule `DPMAX` shows the symbolic interpretation of `dpMax` whose iCDF aligns with the one appearing in [8]. Observe that the return value is tainted. The reason for that relies in the fact that the result, which is one of the responses in `res`, contains no noise—it is rather the process that lead to determining the winning response which has been “noisy.” In this light, no scale of noise nor distribution can be associated to the response—as we did, for instance, with `dpCount`.

To symbolically interpret a sequence of primitives, the analysis gets further split into two cases depending if the first operation to interpret is a transformation or an aggregation, respectively—see Figure A.9b. Rule `SEQ-TRANS` considers the former, where `transform` can be any of the transformation operations in Figure A.4. It simply uses the symbolic value \mathcal{D} to pass it to the continuation k . It can happen that $k \ \mathcal{D}$ does not match (yet) any part of DPella’s API required for our analysis to continue⁷. However, the EDSL nature of DPella makes Haskell’s to reduce $k \ \mathcal{D}$

⁶We ignore the case of `return val :: Query (Value a)` since the definition of `accuracy` is trivial for such case.

⁷For instance, $k \ \mathcal{D} = (\lambda x \rightarrow \text{dpCount } 1 \ x) \ \mathcal{D}$, and thus $((\lambda x \rightarrow \text{dpCount } 1 \ x) \ \mathcal{D}) \rightsquigarrow^* \text{dpCount } 1 \ \mathcal{D}$.

$$\begin{array}{c}
\text{DPCOUNT} \\
\text{dataset} :: \text{Data } s \ r \quad \text{iCDF} = \lambda\beta \rightarrow \log\left(\frac{1}{\beta}\right) \cdot s \cdot \frac{1}{\epsilon} \quad t \text{ fresh} \\
\hline
\text{dpCount} \in \text{dataset} \triangleright \mathcal{S}\left[\text{iCDF}, s \cdot \frac{1}{\epsilon}, \{t\}\right]
\end{array}$$

$$\begin{array}{c}
\text{DPMAX} \\
\text{dataset} :: \text{Data } l \ r \quad \text{iCDF} = \lambda\beta \rightarrow \frac{4}{\epsilon} \cdot \log\left(\frac{\text{length res}}{\beta}\right) \\
\hline
\text{dpMax} \in \text{res vote ds} \triangleright \mathcal{S}[\text{iCDF}, 0, \emptyset]
\end{array}$$

(a) DP-queries

$$\begin{array}{c}
\text{SEQ-TRANS} \\
k \mathcal{D} \rightsquigarrow^* \text{next} \quad \text{next} \triangleright \mathcal{S}[\text{iCDF}, s, \text{ts}] \\
\hline
\text{transform} >>= k \triangleright \mathcal{S}[\text{iCDF}, s, \text{ts}]
\end{array}$$

$$\begin{array}{c}
\text{SEQ-QUERY} \\
\text{query} \triangleright \mathcal{S}[\text{iCDF}_q, s_q, \text{ts}_q] \quad k (\mathcal{S}[\text{iCDF}_q, s_q, \text{ts}_q]) \rightsquigarrow^* \text{next} \quad \text{next} \triangleright \mathcal{S}[\text{iCDF}, s, \text{ts}] \\
\hline
\text{query} >>= k \triangleright \mathcal{S}[\text{iCDF}, s, \text{ts}]
\end{array}$$

(b) Sequential traversal

$$\begin{array}{c}
\text{SEQ-PART} \\
(m \ j \ \mathcal{D} \rightsquigarrow^* \text{next}_j)_{j \in \text{dom}(m)} \quad (\text{next}_j \triangleright \mathcal{S}[\text{iCDF}_j, s_j, \text{ts}_j])_{j \in \text{dom}(m)} \\
m' = (j \mapsto \mathcal{S}[\text{iCDF}_j, s_j, \text{ts}_j])_{j \in \text{dom}(m)} \quad k \ m' \rightsquigarrow^* \text{next} \quad \text{next} \triangleright \mathcal{S}[\text{iCDF}, s, \text{ts}] \\
\hline
\text{dpPart sel dataset } m >>= k \triangleright \mathcal{S}[\text{iCDF}, s, \text{ts}]
\end{array}$$

(c) Accuracy calculation when partitioning data

Figure A.9: Accuracy analysis implemented by `accuracy`

to the `next` primitive to be considered, which we capture as $k \mathcal{D} \rightsquigarrow^* \text{next}$ —and we know that it will occur thanks to type preservation. We represent \rightsquigarrow (\rightsquigarrow^*) to pure reduction(s) in the host language like function application, pair projections, list comprehension, etc. The analysis then continues symbolically interpreting the `next` yield instruction. Rule `SEQ-QUERY` computes the corresponding symbolic value for the aggregation query. The symbolic value is then passed to the continuation, and the analysis continues with the `next` yield instruction.

Rule `SEQ-PART` shows the symbolic interpretation of `dpPart`. The argument $m :: \text{Map } k (\text{Data } s \ r \rightarrow \text{Query } (\text{Value } a))$ describes the queries to execute once given the corresponding bins. Since these queries produce values, we need to symbolically interpret each of them to obtain their accuracy estimations. The rule applies each of those queries to a symbolic dataset $(m \ j \ \mathcal{D})^8$. The symbolic values yield by each bin

⁸For simplicity, we assume that maps are implemented as functions

$$\begin{array}{c}
\text{UNION-BOUND} \\
\frac{v_j = \mathcal{S}[\text{iCDF}_j, s_j, \text{ts}_j] \quad \alpha_j = \text{iCDF}_j\left(\frac{b}{n}\right) \quad \text{iCDF} = \lambda\beta \rightarrow \sum_{j=1}^n \alpha_j}{\text{ub}[v_1, v_2, \dots, v_n] \rightsquigarrow \text{iCDF}}
\\[10pt]
\text{CHERNOFF-BOUND} \\
\frac{v_j = \mathcal{S}[\text{iCDF}_j, s_j, \text{ts}_j] \quad s_M = \max\{s_j\}_{j=1\dots n} \quad \nu = \max\left\{\sqrt{\sum_{j=1}^n s_j^2}, s_M \cdot \sqrt{\ln\left(\frac{2}{\beta}\right)}\right\} + 0.0001 \quad \text{iCDF} = \lambda\beta \rightarrow \nu \cdot \sqrt{8 \cdot \ln\left(\frac{2}{\beta}\right)}}{\text{cb}[v_1, v_2, \dots, v_n] \rightsquigarrow \text{iCDF}}
\\[10pt]
\text{ADD-UNION} \\
\frac{(\exists j. \text{ts}_j = \emptyset) \vee \bigcap_{j=1\dots n} \text{ts}_j \neq \emptyset}{\text{add}[v_1, v_2, \dots, v_n] \rightsquigarrow \mathcal{S}[\text{ub}[v_1, v_2, \dots, v_n], 0, \emptyset]}
\\[10pt]
\text{ADD-CHERNOFF-UNION} \\
\frac{\bigcap_{j=1\dots n} \text{ts}_j = \emptyset \quad v_j = \mathcal{S}[\text{iCDF}_j, s_j, \text{ts}_j] \quad (\forall j. \text{ts}_j \neq \emptyset) \quad \text{iCDF} = \lambda\beta \rightarrow \min(\text{ub}[v_1, v_2, \dots, v_n] \beta) (\text{cb}[v_1, v_2, \dots, v_n] \beta)}{\text{add}[v_1, v_2, \dots, v_n] \rightsquigarrow \mathcal{S}[\text{iCDF}, 0, \emptyset]}
\end{array}$$

Figure A.10: Calculation of concentration bounds

are collected into the mapping m' , which is then passed to continuation k in order to continue the analysis on the next yield instruction.

Concentration Bounds

Figure A.10 shows the part of our analysis responsible to apply concentration bounds. Rules UNION-BOUND and CHERNOFF-BOUND define pure functions (reduction \rightsquigarrow) which produces the concentration bounds as described in Definitions A.5 and A.6, respectively. We define the function `add` based on two cases. Rule ADD-UNION produces a symbolic value with a iCDF generated by the union bound (expressed as $\text{ub}[v_1, v_2, \dots, v_n]$). The symbolic value is tainted, which is denoted by the empty tags (\emptyset) . The scale `0` denotes that the scale of the noise and its distribution is unknown—adding Laplace distributions do not yield a Laplace distribution. (However, the situation is different with Gaussians, see Section A.5.3) This rule gets exercised when either the list of symbolic values contains a tainted one $(\exists j. \text{ts}_j = \emptyset)$ or have not been independently generated $(\bigcap_{j=1\dots n} \text{ts}_j \neq \emptyset)$. Differently, ADD-CHERNOFF-UNION produces a symbolic value with a iCDF which chooses the minimum error estimation between union and Chernoff bound for a given β —sometimes union bound provides tighter estimations when aggregating few noisy-values (recall Figure A.7). This rule triggers when all the values are untainted $(\forall j. \text{ts}_j \neq \emptyset)$ and independently generated $(\bigcap_{j=1\dots n} \text{ts}_j = \emptyset)$. At a first glance, one could believe that it would be enough to use the scale of the noise to track when values are untainted, i.e., if the scale is different from 0, then the value is untainted. Unfortunately, this design choice is unsound: it will classify adding a variable twice as an independent sum: `do $x \leftarrow \text{dpCount } \epsilon \text{ ds}; \text{return } (\text{add}[x, x])$` . It is also possible to consider various ways to add symbolic values to boost accuracy. We could easily

$$\begin{array}{c}
\text{NORM-INF} \\
\frac{v_j = \mathcal{S}[\text{iCDF}_j, s_j, ts_j] \quad \text{iCDF} = \lambda\beta \rightarrow \max \{ |\text{iCDF}_j(\frac{\beta}{n})| \}_{j=1\dots n}}{\text{norm}_\infty [v_1, v_2, \dots, v_n] \rightsquigarrow \mathcal{S}[\text{iCDF}, 0, \emptyset]} \\
\\
\text{NORM-1} \\
\frac{v_j = \mathcal{S}[\text{iCDF}_j, s_j, ts_j] \quad \text{iCDF} = \lambda\beta \rightarrow \sum_{j=1}^n |\text{iCDF}_j(\frac{\beta}{n})|}{\text{norm}_1 [v_1, v_2, \dots, v_n] \rightsquigarrow \mathcal{S}[\text{iCDF}, 0, \emptyset]}
\end{array}$$

Figure A.11: Calculation of norms

write a pre-processing function which, for instance, firstly partitions the arguments into subset of independently generated values, applies `add` to them (thus triggering `ADD-CHERNOFF-UNION`), and finally applies `add` to the obtained results (thus triggering `ADD-UNION`). The implementation of DPella enables to write such functions in a few lines of code.

Norms calculation

Figure A.11 shows our static analysis when computing `norm∞` and `norm1`, respectively. There is nothing special about the rules except to note that the results are symbolic values which are tainted. The reason for that is that norms are designed to condense (in one measure) the error of the list of the arguments. By doing so, it is hard to assign an specific Laplace distribution with sensitivity s to the overall given vector. We simply say that the return symbolic values are tainted—thus they can only be aggregated by `ADD-UNION` in Figure A.10.

A.5.3 Accuracy of Gaussian mechanism

As aforementioned, DPella supports other notions of differential privacy—such as approximate differential privacy—together with the use of the Gaussian mechanism. Specifically, DPella supports a relaxation of the notion of differential privacy known as (ϵ, δ) -DP, formally defined as follow.

Definition A.7 ((ϵ, δ) -Differential Privacy[18]). *A randomized query $\tilde{Q}(\cdot) : \text{db} \rightarrow \mathbb{R}$ satisfies (ϵ, δ) -differential privacy, with $\epsilon, \delta \geq 0$, if and only if for any two datasets D_1 and D_2 in db, which differ in one row, and for every output set $S \subseteq \mathbb{R}$ we have*

$$\Pr[\tilde{Q}(D_1) \in S] \leq e^\epsilon \Pr[\tilde{Q}(D_2) \in S] + \delta \quad (\text{A.6})$$

The main difference between this notion of privacy and the one described in Theorem A.1 is that (ϵ, δ) -DP introduces the probability mass δ that, intuitively, offers a probabilistic notion of privacy loss. More concretely, (ϵ, δ) -DP ensures that for all adjacent datasets, the absolute value of the privacy loss will be bounded by ϵ with probability $1 - \delta$. Observe that when $\delta = 0$, an $(\epsilon, 0)$ -DP query satisfies pure ϵ -DP.

A standard implementation of (ϵ, δ) -DP queries is based on the addition of noise sampled from the Gauss distribution, this is, for $Q : \text{db} \rightarrow \mathbb{R}$ an arbitrary function

$$\begin{array}{c}
\text{DPCOUNT} \\
\frac{\text{dataset} :: \text{Data } \mathbf{s} \ \mathbf{r} \quad \sigma = \sqrt{2 \cdot \log\left(\frac{1.25}{\delta}\right)} \cdot \mathbf{s} \cdot \frac{1}{\epsilon} \quad \text{iCDF} = \lambda\beta \rightarrow \sigma \cdot \sqrt{2 \cdot \log\left(\frac{2}{\beta}\right)} \quad \mathbf{t} \text{ fresh}}{\text{dpCount } \epsilon \text{ dataset } \triangleright S[\text{iCDF}, \sigma^2, \{\mathbf{t}\}]} \\
\\
\text{(a) Aggregations} \\
\\
\text{CHERNOFF-BOUND-GAUSS} \\
\frac{\mathbf{v}_j = S[\text{iCDF}_j, \mathbf{s}_j, \mathbf{ts}_j] \quad \text{iCDF} = \lambda\beta \rightarrow \sqrt{2 \cdot \sum_{j=1}^n \mathbf{s}_j \cdot \log\left(\frac{1}{\beta}\right)}}{\text{cb } [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] \rightsquigarrow \text{iCDF}} \\
\\
\text{ADD-CHERNOFF-UNION} \\
\frac{\bigcap_{j=1 \dots n} \mathbf{ts}_j = \emptyset \quad \mathbf{v}_j = S[\text{iCDF}_j, \mathbf{s}_j, \mathbf{ts}_j] \quad (\forall j. \mathbf{ts}_j \neq \emptyset) \quad \text{iCDF} = \lambda\beta \rightarrow \min(\text{ub } [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] \ \beta)(\text{cb } [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] \ \beta)}{\text{add } [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] \rightsquigarrow S[\text{iCDF}, \sum_{j=1}^n \mathbf{s}_j, \bigcup_{j=1 \dots n} \mathbf{ts}_j]} \\
\\
\text{(b) Concentration bounds}
\end{array}$$

Figure A.12: Accuracy analysis for Gaussian mechanism

with sensitivity Δ_Q (as described in Definition A.2) the Gaussian mechanism with parameter σ adds noise scaled to $\mathcal{N}(0, \sigma^2)$ to its output. When the noise to be added is calibrated in terms of ϵ , δ , and Δ_Q , the Gaussian mechanism satisfies (ϵ, δ) -DP as stated on the following theorem.

Theorem A.4 (Gaussian Mechanism [2]). *For any $\epsilon, \delta \in (0, 1)$, the Gaussian output perturbation mechanism with standard deviation $\sigma = \sqrt{2 \cdot \log\left(\frac{1.25}{\delta}\right)} \cdot \frac{\Delta_Q}{\epsilon}$ is (ϵ, δ) -differentially private*

Similarly as with the Laplace mechanism, to provide bound estimates on the errors caused by the addition of Gaussian noise, DPella keeps track of Gauss' *inverse Cumulative Distribution Function* (iCDF). By following the general form of accuracy introduced in Definition A.2, we have that:

Definition A.8 (Accuracy for the Gaussian mechanism). *Given a randomized query $\tilde{Q}(\cdot) : \text{db} \rightarrow \mathbb{R}$ implemented with the Gaussian mechanism as previously described, then*

$$\Pr \left[|\tilde{Q}(D) - Q(D)| > \sigma \cdot \sqrt{2 \cdot \log\left(\frac{2}{\beta}\right)} \right] \leq \beta \quad (\text{A.7})$$

where the iCDF to be stored by DPella refers to the function $\lambda\beta \rightarrow \sigma \cdot \sqrt{2 \cdot \log\left(\frac{2}{\beta}\right)}$.

From an implementation standpoint, adding the Gaussian mechanism to our framework does not alter significantly the presented primitives, and, in particular, privacy's preservation remains (almost) unchanged. The most significant changes can be seen when calculating the accuracy of aggregations and their combinations.

The symbolic interpretation of aggregations is updated accordingly to keep track of Gauss' iCDF, as well as, its respective noise scale determined by σ^2 as depicted

in Figure A.12a for the case of `dpCount`. Additionally, Figure A.12b shows how concentration bounds are applied for the case of the Gaussian mechanism—`UNION-BOUND` and `ADD-UNION` are omitted since they are the same as the ones in Figure A.10. In general, the accuracy analysis for addition of aggregations follows the one presented previously for the Laplace mechanism. The main difference is seen when adding independent values. In this case, we use the well-known fact of the addition of independent normally distributed random variables are also normally distributed. This means that after executing the `ADD-CHERNOFF-UNION` we do not lose information about the distribution of our results as we used to do under the Laplacian setting. This effect can be seen in the generated symbolic value $S[\text{iCDF}, \sum_{j=1}^n s_j, \bigcup_{j=1\dots n} \text{ts}_j]$ where $\sum_{j=1}^n s_j$ indicates that the variance of the new value is calculated as the addition of the variances of the components being added, and $\bigcup_{j=1\dots n} \text{ts}_j$ indicates that the new value is statistically dependent on the involved values.

```

1 totalCountG :: Query (Value Double)
2 totalCountG = do
3   v1 ← dpCount (0.3 , 1e-5) ds1
4   v2 ← dpCount (0.25, 1e-5) ds2
5   ...
6   v100 ← dpCount (0.5 , 1e-3) ds100
7   let h1 = add [v1, v2, ..., v50]
8   let h2 = add [v51, v52, ..., v100]
9   return (add [h1, h2])

```

Figure A.13: Combination in batches

This is an useful feature when combining queries in batches, for instance, Figure A.13 shows the query plan `totalCountG` that adds the results of hundred queries—using Gaussian `dpCount` that takes as input the tuple (ϵ, δ) and the dataset—similar to the one presented in Figure A.8, but it does so by adding the first half of the queries (line 7), then the second half (line 8), and finally returning the addition of the two halves (line 9). How will DPella calculate the theoretical error of `totalCountG`?

Observe that h_1 and h_2 are constructed as combinations of untainted values, meaning that when performing the additions at lines 7-8, the Chernoff bound could be triggered. More importantly, DPella still have information about their distribution. Furthermore, h_1 and h_2 are statistically independent (they do not share sub-queries), so when computing their addition at line 9, Chernoff bound could also be triggered, this could not have been possible under the Laplace mechanism, since once a value is calculated as a combination of values, their distribution becomes unknown and only union bound could be applied. In this sense, the Gaussian mechanism might yield tighter error bounds when dealing with queries that are created in batches, specially when the number of batches is big enough to trigger the use of the Chernoff bound.

A.6 Case studies

In this section, we will discuss the advantages and limitations of our programming framework. Moreover, we will go in-depth into using DPella to analyze the interplay of privacy and accuracy parameters in hierarchical histograms.

Category	Application	Programs
PINQ-like	CDFs [40]	<code>cdf1</code> , <code>cdf2</code> , <code>cdfSmart</code>
	Term frequency [28]	<code>queryFreq</code> , <code>queriesFreq</code>
	Network analysis [40]	<code>packetSize</code> , <code>portSize</code>
	Cumulative sums [6]	<code>cumulSum1</code> , <code>cumulSum2</code> , <code>cumulSumSmart</code>
Counting queries	Range queries via Identity, Histograms [30], and Wavelet [62]	<code>i_n</code> , <code>h_n</code> , <code>y_n</code>

Table A.1: Implemented literature examples

A.6.1 DPella expressiveness

First, we start by exploring the expressiveness of DPella. For this, we have built several analyses found in the DP literature—see Table A.1—which we classify into two categories, *PINQ-like queries* and *counting queries*. The former class allows us to compare DPella expressivity with the one of PINQ, while the latter with APEx.

PINQ-like queries We have implemented most of PINQ’s examples [28, 40], such as, different versions of CDFs (sequential, parallel, and hybrid) and network tracing-like analyses (such as determining the frequency a term or several terms have been searched by the users, and computing port’s and packets’ size distribution); additionally, we considered analyses of *cumulative sums* [6]—which are queries that share some commonalities with CDFs. The interest over differentially private CDFs and cumulative partial sums applications rely on the existing several approaches to inject noise, such choices will directly impact the accuracy of our results, and therefore, are ideal to be tested and analyzed in DPella. The structures of these examples follow closely the ones of the CDFs presented in previous sections, which are straightforward implementations. DPella supports these queries naturally since its expressiveness relies on its primitives and, by construction, they follow PINQ’s ones very closely. However, as stated in previous sections, our framework goes a step further and exposes to data analysts the accuracy bound achieved by the specific implementation. This feature allows data analyst to reason about accuracy of the results—without actually executing the query—by varying i) the strategy of the implementation ii) the parameters of the query. For instance, in Section A.3, we have shown how an analyst can inspect the error of a sequential and parallel strategy to compute the CDF of packet lengths. Furthermore, the data analyst can take advantage of DPella being an embedded DSL and write a Haskell function that takes any of the approaches (`cdf1` or `cdf2`) and varies epsilon aiming to certain error tolerance (for a fixed confidence interval), or vice versa. Such a function can be as simple as a brute force analysis or as complex as an heuristic algorithm.

Counting queries To compare our approach with the tool APEx [25], we consider range queries analyses—an specific subclass of counting queries. APEx uses the *matrix mechanism* [35] to compute counting queries. This algorithm answers a

$$\begin{array}{ccc}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
& \mathbf{I}_4 & \mathbf{H}_4 \\
\mathbf{W}_{R_4} & & \mathbf{Y}_4
\end{array}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Figure A.14: Workload of all range queries and query strategies for 4 ranges

set of linear queries (called the *workload*) by calibrating the noise to specific properties of the workload while preserving differential privacy. More in detail, the matrix mechanism uses some *query strategies* as an intermediate device to answer a workload; returning a DP version of the query strategies (obtained using the Laplace or Gaussian mechanism), from which noisy answers of the workload are derived. The matrix mechanism achieves an almost optimal error on counting queries. To achieve such error, the algorithm uses several non-trivial transformations which cannot be implemented easily in terms of other components. APEX implements it as a black-box and we could do the same in DPella (see Section A.9). Instead, in this section we show how DPella can be directly used to answer sets of counting queries using some of the ideas behind the design of the matrix mechanism, and how these answers improve with respect to answering the queries naively, thanks to the use of partition and the Chernoff bound.

To do this, we have implemented several strategies to answer an specific workload \mathbf{W}_R : the set of all range queries over a domain. Figure A.14 illustrates the workload that would be answer for a frequency count of four ranges. The identity strategy \mathbf{I}_4 , represents 4 queries (number of rows) computing the noisy count of each range (number of columns). The hierarchical strategy \mathbf{H}_4 contains seven queries representing a binary hierarchy of sums, while the wavelet strategy \mathbf{Y}_4 contains four queries representing the Haar wavelet matrix.

Our implementation generates noisy counts and any possible combination of them will yield (at least) the same error as using strategy \mathbf{I}_4 . In other words, the more accurate answer for \mathbf{W}_R will be yield by the identity strategy. This is not unexpected, since in order to use the other queries strategies more efficiently we would need transformations similar to the ones used in the matrix mechanism.

Figure A.15 exposes the error of answering each range query (i.e., each row) in \mathbf{W}_R with strategy \mathbf{I}_n and $n = 512$. While we use the same kind of plot, this error cannot be directly compared with the one shown in Figure 7 of [35], since we use a different error metrics: (α, β) -accuracy vs MSE. Nonetheless, we share the tendency of having lower error on small ranges and significant error on large ranges. Now, since the noisy values that will be added (using the function `add`) are statistically in-

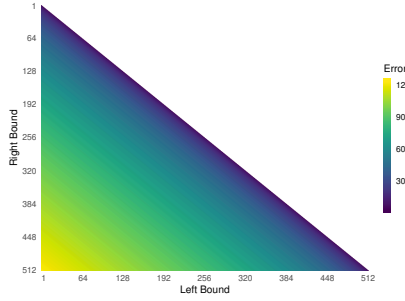


Figure A.15: Error of each range query in \mathbf{W}_R using strategy \mathbf{I}_n with $n = 512$, $\epsilon = 1$, and $\beta = 0.05$

```

1 alert ::  $\epsilon \rightarrow \text{Data } 1 \text{ RowV} \rightarrow \text{Query } (\text{Value } [\text{Phone}])$ 
2 alert eps ds = do
3   atRisk  $\leftarrow \text{dpMax } \text{eps allZIPs zip ds}$ 
4   return (useIndex getPhone atRisk)
5   where
6     getPhone :: ZIP  $\rightarrow [\text{Phone}]$ 
7     getPhone z = [snd info | info  $\leftarrow$  contact, fst info  $\equiv$  z]

```

Figure A.16: Using `dpMax`

dependent, we can use the Chernoff bound to show that the error is approximately $\mathcal{O}(\sqrt{n})$ for each range query, and a maximum error of $\mathcal{O}(\sqrt{n \log n})$ for answering any query in \mathbf{W}_R . If we compare our maximum error $\mathcal{O}(\sqrt{n \log n})$ with the one of the matrix mechanism based on the identity strategy $\mathcal{O}(n/\epsilon^2)$, it becomes evident how Chernoff bound is useful to provide tighter accuracy bounds. Unfortunately, as previously stated, the error of strategies \mathbf{H}_n and \mathbf{Y}_n in DPella is not better than the one of the strategy \mathbf{I}_n , so we cannot reach the same accuracy the matrix mechanism achieves with these strategies (see Figure 7 of [35]). This limitation can be addressed by leveraging the fact that DPella is a programming framework that could be *extended* by adding the matrix mechanism—and some other features—as black-box primitives.

Black-box primitives To demonstrate the effects of adding black-box operators in DPella, let us consider a rather simple query using primitive `dpMax`. Suppose there is a highly contagious virus spreading in a state. To reduce this virus’s rapid spreading, one might want to alert the population where there are more cases of infections so they can quarantine. In this scenario, we will consider two sources of information. One that is private, containing information about infected patients such as their identity number and ZIP code (this is, `data RowV = V { id :: String, zip :: ZIP }`); and other that is public, such as the phone numbers of people living in each ZIP code (referred to as `contact :: [(ZIP, Phone)]`).

With this information at hand, we implement query `alert` (Figure A.16). Function `alert` uses primitive `dpMax` (line 3) to access the private dataset of infected patients and return the ZIP code with more infection cases, which will determine which zone is at risk. Then we obtain the phone number of all residents of the area using `getPhone` function (lines 6-7). Additionally, in line 4, we function `useIndex` to access ZIP code enclosed in `atRisk` to latter be used by function `getPhone`.

Function `useIndex` is then a new combinator with type `useIndex :: (a → b) → Value a → Value b` that has been introduced to be applied only to values generated with `dpMax`. The insight behind its implementation relies on the fact that the output of a `dpMax` computation does not contain noise; therefore, applying any function to its result should not affect its interpretation of accuracy.

From this example, we expect to illustrate the convenience of adding differentially-private algorithms as black-boxes in DPella (such as `dpMax`) and their relatively smooth integration into the system once their accuracy estimation has been determined. Ultimately, it highlights that integrating these primitives usually cascades into defining new combinators (as `useIndex`) to further manipulate their outputs.

A.6.2 Privacy and accuracy trade-off analysis

We study histograms with certain hierarchical structures (commonly seen in Census Bureaus analyses) where different accuracy requirements are imposed per level and where varying one privacy or accuracy parameter can have a *cascade impact* on the privacy or accuracy of others. We consider the scenario where we would like to generate histograms from the Adult database⁹ to perform studies on gender balance. The information that we need to mine is not only an histogram of the genders (for simplicity, just male and female) but also how the gender distributes over age, and within that, how age distributes over nationality—thus exposing a hierarchical structure of three levels.

Our first approach is depicted in Figure A.17a, where `hierarchical1` generates three histograms with different levels of details. This query puts together the results produced by queries `byGen`, `byGenAge`, and `byGenAgeNationality` where each query generates an histogram of the specified set of attributes. Observe that these sub-queries are called with potentially different epsilons, namely ϵ_1 , ϵ_2 , and ϵ_3 , then under sequential composition, we expect `hierarchical1` to be $\epsilon_1 + \epsilon_2 + \epsilon_3$ -differentially private.

We proceed to explore the possibilities to tune the privacy and accuracy parameters to our needs. In this case, we want a confidence of 95% for accuracy, i.e., $\beta = 0.05$, with a total budget of 3 ($\epsilon = 3$). We could manually try to take the budget $\epsilon = 3$ and distribute it to the different histograms in many different ways and analyze the implication for accuracy by calling `accuracy` on each sub-query. Instead, we write a small (simple, brute force) optimizer in Haskell that splits the budget uniformly among the queries, i.e., $\epsilon_1 = 1$, $\epsilon_2 = 1$, and $\epsilon_3 = 1$, and tries to find the minimum epsilon that meets the accuracy demands per histogram. In other words, we are interested in minimizing the *privacy loss* at each level bounding the max-

⁹<https://archive.ics.uci.edu/ml/datasets/adult>

```

1 hierarchical1 [e1, e2, e3] dat = do
2   -- h1 :: Map Gen (Value Double)
3   -- h2 :: Map (Gen, Age) (Value Double)
4   -- h3 :: Map (Gen, Age, Nationality) (Value Double)
5   h1 ← byGen      e1 dat
6   h2 ← byGenAge   e2 dat
7   h3 ← byGenAgeNat e3 dat
8   return (h1, h2, h3)

```

(a) Approach I: distribute budget among levels

```

9 hierarchical2 e dat = do
10  h3 ← byGenAgeNat e dat
11  h2 ← level2 h3
12  h1 ← level1 h3
13  return (h1, h2, h3)

```

(b) Approach II: query most detailed level

Figure A.17: Implementation of hierarchical histograms

imum accepted error. The optimizer essentially adjusts the different epsilons and calls `accuracy` during the minimization process. To ensure termination, the optimizer aborts after a fixed number of calls to `accuracy`, or when the local budget `e_i` is exhausted.

Table A.2 shows some of our findings. The first row shows what happens when we impose an error of 100 at every level of detail, i.e., each bar in all the histograms could be at most $+/- 100$ off. Then, we only need to spend a little part of our budget—the optimizer finds the minimum epsilons that adheres to the accuracy constraints. Instead, the second row shows that if we ask to be *gradually* more accurate on more detailed histograms, then the optimizer could fulfill the first two demands

Histogram	α tolerance	Status	ϵ	α
byGen	100	✓	0.06	61.48
byGenAge	100	✓	0.06	96.13
byGenAgeNat	100	✓	0.11	85.74
byGen	10	✓	0.41	8.99
byGenAge	50	✓	0.16	36.05
byGenAgeNat	5	✗ MaxBud	1	9.43
byGen	5	✓	0.76	4.85
byGenAge	5	✗ MaxBud	1	5.76
byGenAgeNat	10	✓	0.96	9.82

Table A.2: Budgeting with α tolerances, $\beta = 0.05$, & total $\epsilon = 3$

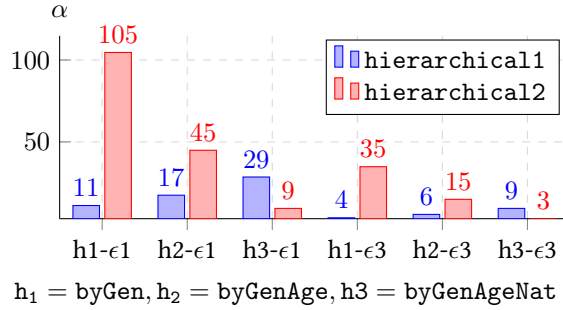


Figure A.18: `hierarchical1` vs. `hierarchical2`

and aborted on the most detailed histogram (`byGenAgeNat`) since it could not find an epsilon that fulfills that requirement—the best we can do is spending all the budget and obtain an error bound of 9.43. Finally, the last row shows what happens if we want *gradually* tighter error bounds on the less detailed histograms. In this case, the middle layer can be “almost” fulfilled by expending all the budget and obtaining an error bound of 5.76 instead of 5. While the results from Table A.2 could be acceptable for some data analysts, they might not be for others.

We propose an alternative manner to implement the same query which consists on spending privacy budget *only* for the most detailed histogram. As shown in Figure A.17b, this new approach spends all the budget ϵ on computing $h_3 \leftarrow \text{byGenAgeNat } \epsilon \text{ dat}$. Subsequently, the algorithm builds the other histograms based on the information extracted from the most detailed one. For that, we add the noisy values of h_3 (using helper functions `level2` and `level1`) creating the rest of the histograms representing the Cartesian products of gender and age, and gender, respectively. This methodology will use `add` and `norm∞` to compute the derived histograms, and therefore will not consume more privacy budget. Observe that the query proceeds in a bottom-up fashion, i.e., it starts with the most detailed histogram and finishes with the less detailed one. Now that we have two implementations, which one is better? Which one yields the better trade-offs between privacy and accuracy? Figure A.18 shows the accuracy of the different level of histograms, i.e., h_1 , h_2 , and h_3 , when fixing $\beta = 0.05$ and a global budget of $\epsilon = 1$ ($h1-\epsilon1$, $h2-\epsilon2$, and $h3-\epsilon3$) and $\epsilon = 3$ ($h1-\epsilon3$, $h2-\epsilon3$, and $h3-\epsilon3$)—we obtained all this information by running repetitively the function `accuracy`. From the graphics, we can infer that splitting the privacy budget per level often yields more accurate histograms. However, observe the exception at the most detailed histogram h_3 : as expected `hierarchical1` will use just one third of the budget while `hierarchical2` uses all the of it, hence the first approach will return noisier count.

A.6.3 K-way marginal queries on synthetic data

When compared with (non-compositional) approaches to estimating accuracy based on synthetic or public data, such as [29], the static analysis of DPella can be used in

```

1  -- Perform all 3-way combinations up to attribute dim
2  allChecks ::  $\epsilon \rightarrow \text{Int} \rightarrow \text{Data s BinR} \rightarrow [\text{Query (Value Double)}]$ 
3  allChecks localEps dim db = do
4    (i, j, k) ← combinatory (dim-1) 3
5    let allOne r = (r !! i) ≡ (r !! j) ≡ (r !! k) ≡ 1
6    return (do tab ← dpWhere allOne db
7              dpCount localEps tab)

8  -- Compute k-way marginals
9  threeMarginal ::  $\epsilon \rightarrow \text{Int} \rightarrow \text{Data s BinR} \rightarrow \text{Query (Value [Double])}$ 
10 threeMarginal localEps dim db = do
11   checks ← sequence (allChecks localEps dim db)
12   return (norm∞ checks)

```

Figure A.19: K-way marginal implementation

a complimentary manner to quickly (and precisely) estimate privacy and accuracy for a wide range of simple queries. There are certain kinds of queries where it is more convenient to use our static analysis than synthetic data for high-dimensional datasets.

As an example, we focus on the problem of releasing, in a differentially private manner, the k -way marginals of a binary dataset $D \in (\{0, 1\}^d)^n$. This is a classical learning problem that has been extensively studied in the DP literature, see [59, 22, 14] among others. A k -marginal query, also called a k -conjunction, returns the count of how many individual records in D have $k < d$ attributes set to certain values. For simplicity, we will work with 3-way marginal queries to compare performance between DPella and using synthetic data. The goal of our analysis is to release all the 3-way marginals of a dataset. This is implemented through the functions depicted in Figure A.19.

Function `allChecks` counts how many records have 3-attributes set to 1. Auxiliary function `combinatory d k` generates k -tuples arising from the combination of indexes $0, 1, \dots, d$ taken k at the time. In our example, the number of generated tuples is $\binom{d}{3}$. For each tuple, `allChecks` filters the rows which have attributes i, j , and k set to 1 (implemented as `dpWhere allOne db`) for then making a noisy count (`dpCount localEps tab`). Lastly, function `threeMarginal` collects the counts for the different considered attributes and places them into a vector (`norm∞ checks`).

We run `threeMarginal` considering a synthetic dataset (`db`) which has only 1 row with all the attributes set to zeros. Setting all the attributes to zero produces that all the counts are 0, thus we can measure the noise on each run and accuracy accordingly. We run `threeMarginal` approx. 1000 times for each dimension to measure the noise magnitude, where we took the $1-\beta$ percentile with $\beta = 0.05$ (as we did in many of our case studies). Observe that we have $\binom{d}{3}$ queries and so $\binom{d}{3}$ independent sources of noise, which need a high number of runs to be well-represented. In general, for this kind of task, one is interested in bounding the max

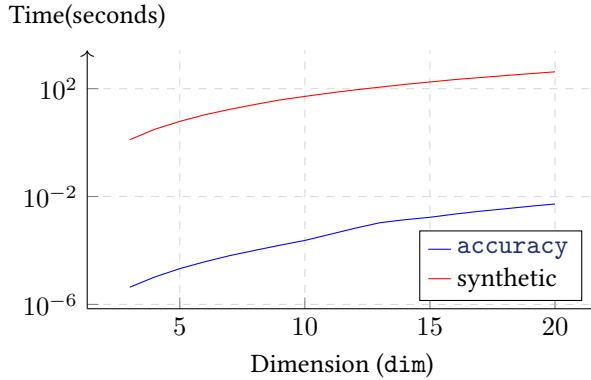


Figure A.20: Performance comparison between `accuracy` (DPella) and estimating errors using synthetic analysis

error that can occur in one of the queries (the ℓ_∞ norm over the output). For this task, the empirical error is well aligned with the theoretical one provided by DPella by calling the function `accuracy`. The latter is computed by taking a union bound over the error of each individual query. For each query we have a tight bound and the union bound gives us a tight bound over the max. However, we observe a significant difference in performance.

Figure A.20 shows (in log scale) the time difference when calculating accuracy by DPella and on synthetic data when the dimension of the dataset increases. Already in low dimension, the difference in performance is many orders of magnitude in favor of DPella—a tendency that does not change when the dimension goes above 20. The main reason for this is that DPella’s static analysis, do not execute the filtering `dpWhere allOne db` (as well as any other transformation, recall Section A.5.2) which an approach based on synthetic data should do many times—in our case 1000 iterations for each dimension. We expect that for more complex tasks this difference is even more evident.

A.7 Testing accuracy

In previous sections we have seen the usefulness of `accuracy` function to inspect queries’ error, reason about the trade-offs of privacy and accuracy, among other perks. It is clear then that providing theoretical bounds over the errors of the implemented queries becomes handy to ease and assist data analysts’ tasks. However, one might argue that having a theoretical bound is as important as producing a measurement of the *tightness* of such calculations. In this section, we focus on the verification of how close DPella’s accuracy calculations are to the real error bounds.

Thanks to DPella’s data independence, we have been able to create the primitive `empiric` that allows analysts to compare the theoretical bound (provided by `accuracy`) against an empirical one. It offers a way to compare DPella’s estimations

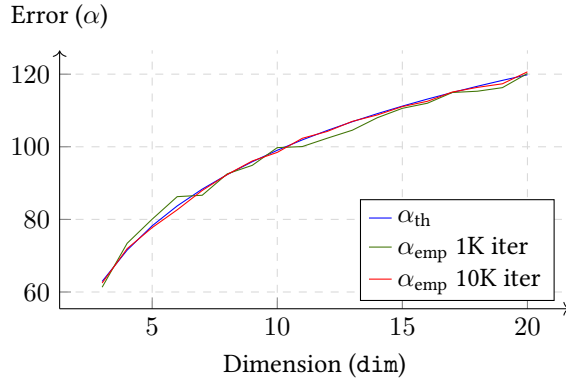


Figure A.21: Results of `empiric` over 3-way marginals

for a query against its empirical error while still preserving the privacy of the data subjects.

The primitive `empiric` is as follows:

`empiric` :: ($\epsilon \rightarrow \text{Data } 1\ r \rightarrow \text{Query (Value a)}$) $\rightarrow \text{Iter} \rightarrow \epsilon \rightarrow \beta \rightarrow \text{IO } \alpha$

Given a query plan (of type $\epsilon \rightarrow \text{Data } 1\ r \rightarrow \text{Query (Value a)}$), a number of iterations (where `Iter` is isomorphic to the type `Int`), a fixed privacy loss ϵ and confidence β ; primitive `empiric` will return the empirical error α_{emp} of the given query using the theoretical error α_{th} provided by `accuracy` with β .

Ideally, α_{emp} should be significantly close to α_{th} . In particular, since `accuracy` yields an *upper bound* of the error, when `empiric` is run multiple times we expect α_{emp} to be less or equal than α_{th} most of the time. The unsatisfiability of this condition indicates that the probability of being above the theoretic error is higher than anticipated, from which we can deduce that DPella’s error estimation is unsound and it does not actually yield an upper bound of the query’s accuracy. On the other hand, if for most of the runs we observe that $\alpha_{emp} \ll \alpha_{th}$, we can infer that DPella’s estimations are loose, indicating that we could either increase the confidence or decrease the error.

The procedure followed by `empiric` is fairly simple. Firstly, it executes the given query as many times as indicated over an *empty dataset*, this process clearly does not involve any sensitive information. However, the attentive reader might have noticed that these executions will allow us to inspect the query’s noise since they will only return the perturbation to be added, this is, we are sampling as many times as iterations from the Laplace distribution (or Gauss, depending on the mechanism) scaled by the sensitivity of the query under consideration. From the samples, we calculate each empirical error α_{emp}^i either applying the absolute value to the i -th sample if the query’s output is a scalar, or the specified norm (i.e., ℓ_∞ , ℓ_2 , ℓ_1 , or rmsd) if the output is multi-dimensional. Then, α_{emp} is computed as the $1 - \beta$ percentile of all α_{emp}^i , where $i = 1, \dots, \text{iter}$.

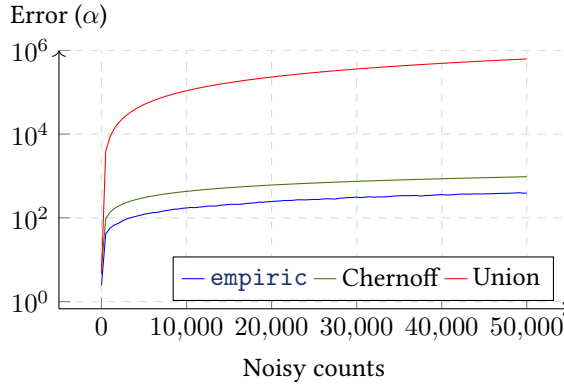


Figure A.22: Results of `empiric` over addition of noisy counts

To illustrate how `empiric` could be used by an analyst, recall the example of the 3-way marginal discussed in the previous section (see Section A.6.3). Previously, we claimed that the empirical error of the function `threeMarginal` from Figure A.19 is well aligned with the theoretical one provided by DPella. This statement can be now verified using `empiric` primitive. For `localEps` = 0.1 and `dim` ranging from 3 to 20, Figure A.21 shows the results of calculating the empirical error of `threeMarginal` with β set to 0.05 and iterating 1000 and 10000 times. From these results, we can conclude that increasing the number of iterations will stabilize the results, making the analyses easier, and that, the empirical error provided for DPella for function `threeMarginal` is indeed very close to the empirical error bound. Moreover, it depicts DPella’s soundness, since in both cases (for 1K and 10K iterations), most of the α_{emp} were below α_{th} ’s line.

We acknowledge that not all configurations of DPella programs will have an accuracy estimation as tight as the one presented above. In particular, one can imagine a scenario where n non-independent noisy values are being added. The theoretical error of such a query will be calculated using the union bound, which has been established [25] to be a loose approximation of the sum’s error. Thus, when comparing the empirical error of these queries against `accuracy`’s projection we should expect a greater discrepancy in favor of the empiric calculation. Under these circumstances, analysts can consider modifying code’s structure to take advantage of Chernoff bound as much as possible or adjust their parameters (ϵ and β).

Since triggering the Chernoff bound produces the best error estimation when adding several (independent) noisy values, one might ask, how tight is this bound? Figure A.22 compares the results of `empiric` over a query adding up to 50000 independent noisy values with β = 0.2 and `localEps` = 1, against the results of `accuracy` triggering the Chernoff bound; additionally the estimations under union bound are shown as a baseline for comparison. Clearly, the union bound’s theoretical errors are several orders of magnitude apart from the empirical errors, showcasing its over-conservative calculations. On the other hand, the Chernoff bound provides a much tighter bound that grows by a constant factor w.r.t. the empirical

error—the reader should keep in mind that the log scale might make it difficult to appreciate this claim; still, it could be quickly confirmed by analyzing the ratio between the two errors.

We argue that DPella’s estimations can be seen as a *quick* initial step into inspecting queries’ accuracy that could be further complemented by other techniques of error estimation—integration of such algorithms are left as future work. DPella’s analyses are particularly useful when dealing with high-dimensional or more complex tasks since—as shown previously—its estimations do not have computational overheads.

A.8 API generalization

Up to this point, we have seen DPella as a framework where data analysts can implement their differentially private consults using either the Laplace or the Gaussian mechanism, but not both simultaneously. However, there might be cases where analysts would want to combine queries using Laplace and Gaussian noise. DPella is designed to allow programmers to mix results from different mechanisms as long as they are implemented under the same privacy notion; for instance, only results from mechanisms deploying approximate-DP can be combined within themselves. To support this behavior, DPella’s computations are generalized by their privacy notion, for which we introduce the abstract data types presented in Figure A.23.

Values of type `Query p` represent computations yielding outputs of type `a` under the privacy notion `p`. Variable `p` can be instantiated to types `PureDP` or `AproxDP`, thus, a term of type `q :: Query Pure (Value Double)` is a pure differentially private computation whose output has type `Value Double`. As established in Section A.4, type `Query p` is a monad, and then sequencing queries is done through the *bind* function. Hence, to enforce combination of queries *only* withing the same privacy notion, the type of function `(>>=)` changes to:

$$(>>=) :: \text{Query } p \ a \rightarrow (a \rightarrow \text{Query } p \ b) \rightarrow \text{Query } p \ b$$

where, all the computations involved must share the same privacy notion `p`. This restriction ensures that the principles of composition can be properly applied when combining queries.

-- Computations	-- Privacy parameters
<code>data Query p a</code>	<code>data Input p m</code>
-- Privacy notions	-- Mechanisms
<code>data PureDP</code>	<code>data Laplace</code>
<code>data AproxDP</code>	<code>data Gauss</code>

Figure A.23: New types for DPellas’s generalization

```

dpCount :: (Stb s, PrivN p, Mech p m) ⇒ Input p m → Data s r
        → Query p (Value Double)
dpSum   :: (Stb s, PrivN p, Mech p m) ⇒ Input p m → Range a b
        → (r → Double) → Data s r → Query p (Value Double)
dpAvg    :: (Stb s, PrivN p, Mech p m) ⇒ Input p m → Range a b
        → (r → Double) → Data s r → Query p (Value Double)
dpMax    :: (Eq a, PrivN p, Mech p m) ⇒ Input p m → Responses a
        → (r → a) → Data 1 r → Query p (Value a)

```

Figure A.24: Updated aggregations

In order to determine which mechanism (or source of noise) will be used in a computation, aggregations are updated to take an argument of type `Input p m` (instead of ϵ or (ϵ, δ)) as depicted in Figure A.24. Type variable `p` still refers to the privacy notion, while `m` indicates which mechanism should be used to ensure p-DP. Variable `m` can be instantiated to types `Laplace` or `Gauss`. Hence, a term of type `arg :: Input AproxDP Gauss` represents an input for a computation guaranteeing approximate-DP using the Gaussian mechanism. The introduction of type `Input p m` allows us to refer to our mechanisms' arguments without specifying them directly. However, when privacy notion and mechanism are chosen, the input gets concretized to either ϵ or (ϵ, δ) . More precisely, the implementation of `Input PureDP Laplace` is isomorphic to ϵ while `Input AproxDP Gauss` and `Input AproxDP Laplace` are isomorphic to (ϵ, δ) .

Lastly, new type constraints `PrivN p` and `Mech p m` are introduced in all aggregations (recall Figure A.24) to avoid invalid combinations of `p` and `m`, e.g., `q :: Query Bool a` and `arg :: Input PureDP Gauss`.

With this new interface analysts can implement generic programs without specifying which mechanism (and privacy notion) will be used during its execution, these computations can be later used to instantiate specific queries. For instance, Figure A.25 presents functions `hist` (lines 1-6) which creates a histogram of ages taking as input a list of bins, a selector function `f`, the general input of the mechanism `arg` and the dataset. This function filters the given dataset accordingly to the selector functor (line 4), partitions the data into the bins and performs a noisy count on each partition (line 5), and finally it returns a list with all noisy counts (line 6). Later, the analyst can call `hist` to define a noisy query under approximate-DP, called `mixHist`, returning a histogram where some bins are computed using the Laplace mechanism (lines 9 and 11-12), while others use the Gaussian mechanism (lines 10 and 13-14). Observe that we use type applications, such as `In @AproxDP @Laplace (...)` and `In @AproxDP @Gauss (...)`, to specify which privacy notion and mechanism should be used.

A.8.1 Implementation and Accuracy estimations

The accuracy analysis of generalized computations follow closely the one defined for the Laplace and Gaussian mechanism. The main difference is that symbolic values

```

1 hist :: (Stb s, PrivN p, Mech p m) => [Age] -> (Age -> Bool) -> Input p m
2   -> Data s Age -> Query p [Value Double]
3 hist bins f arg dataset = do
4   tab  <- dpWhere f dataset
5   parts <- dpPartRepeat (dpCount arg) bins assignBin tab
6   return (Map.elems parts)

7 mixHist :: Query AproxDP (Value [Double])
8 mixHist = do
9   let binsLap   = [5, 10, 15, 20, 25]
10      binsGauss = [50, 55, 60, 65]
11   lapHist <-
12     hist binsLap (<= 25) (In @AproxDP @Laplace (0.25, 1e-3)) ages
13   gaussHist <-
14     hist binsGauss (>= 50) (In @AproxDP @Gauss (0.5, 1e-3)) ages
15   return (norm∞ (lapHist ++ gaussHist))

```

Figure A.25: Histogram using Laplace and Gauss mechanism

$S[\text{iCDF}, d, \text{ts}, \eta]$ now keep track of an extra value, η , representing the distribution from where the noise is drawn. Values of η are limited to L for Laplace, G for Gauss, and U for unknown distributions when noisy values are combined.

The interpretation of aggregations can be summarized by inspecting `dpCount`. Rule DPCOUNT in Figure A.26 shows that the value of η is determined by the distribution indicated at the type of input `arg`, i.e., $\eta = m$. Internally, type Laplace gets translated into the value L and type Gauss into the value G. With this information function `noiseScale` computes the scale of Laplace (following Theorem A.1) or the variance of Gauss (according to Theorem A.4) depending on the value of η . Similarly, function `errorDist` returns the iCDF of the corresponding distribution.

Major changes occur on the interpretation of combinator `add` since now the list of values to be added are potentially mixed w.r.t their distribution. If not done carefully, computing the error of such addition will rarely trigger the Chernoff bound. To maximize the chances of using Chernoff bound rule ADD-CHERNOFF-UNION illustrates how the static analysis splits the symbolic values v_j into three disjoint sets: values V_L are independent Laplacian, values V_G are independent Gaussians, and values V_U are either non-independent or their distribution is unknown. Each of these groups of values will be added between them to latter be used as partial sums for the final result. The resulting value of the partial sum for values in V_L and V_G will have an error estimation computed as the minimum between union and Chernoff bound, this is, $\text{iCDF}_d = \lambda\beta \rightarrow \min(\text{ub } V_d \beta) (\text{cb } V_d \beta)$ for $d \in \{L, G\}$. Since values grouped in V_U do not have an associated Chernoff bound, the iCDF of their addition will be determined by union bound $\text{iCDF}_U = \text{ub } V_U$. Lastly, the iCDF of the addition of all values v_1, v_2, \dots, v_n is computed as the union bound between the values from V_L , V_G , and V_U . This computation is done in such a way that if all values

$$\begin{array}{c}
\text{DPCOUNT} \\
\text{dataset} :: \text{Data } s \ r \quad \text{arg} :: \text{Input } p \ m \quad \eta = m \quad \text{sc} = \text{noiseScale } (s, \text{arg}, \eta) \\
\text{iCDF} = \lambda\beta \rightarrow \text{errorDist } (\beta, s, \text{arg}, \text{sc}, \eta) \quad t \text{ fresh} \\
\hline
\text{dpCount } \text{arg } \text{dataset} \triangleright \mathcal{S}[\text{iCDF}, \text{sc}, \{t\}, \eta] \\
\\
\text{ADD-CHERNOFF-UNION} \\
v_j = \mathcal{S}[\text{iCDF}_j, s_j, ts_j, \eta_j] \quad d \in \{L, G\} \quad V_d = \{v_i \mid \eta_i = d \wedge \bigcap_{i=1 \dots} ts_i = \emptyset\} \\
V_u = v_j \notin \{V_L \cup V_G\} \quad \text{iCDF}_d = \lambda\beta \rightarrow \min(\text{ub } V_d \beta) (\text{cb } V_d \beta) \\
\text{iCDF}_u = \text{ub } V_u \quad (s, ts, \eta) = \text{track } (s_j, ts_j, \eta_j) \\
\hline
\text{add } [v_1, v_2, \dots, v_n] \rightsquigarrow \mathcal{S}[\text{ub } [V_L, V_G, V_u], s, ts, \eta]
\end{array}$$

Figure A.26: Accuracy analysis for mixed mechanisms

v_j come from the same distribution d , the final iCDF will be the same as presented in Figure A.10 for Laplace and Figure A.12b for Gauss.

To determine the values of s , ts , and η function `track` checks if all values are untainted ($\forall j. ts_j \neq \emptyset$), independent ($\bigcap_{j=1 \dots n} ts_j = \emptyset$) and Gaussian ($\forall j. \eta_j = G$), if that is the case then $s = \sum_{j=1}^n s_j$, $ts = \bigcup_{j=1 \dots n} ts_j$, and $\eta = G$ as done previously with the Gaussian mechanism; otherwise $s = 0$, $ts = \emptyset$, and $\eta = U$.

Evidently, combinator `add`'s new behavior will likely yield tighter bounds compared to our previous version since now it procures triggering Chernoff bound on intermediate stages. One can imagine other optimizations over the interpretation of `add` to postpone the tainting of values as much as possible. For example, nested additions such as `add [add [v1, v2], v3, v4, v5]` could be flattened as `add [v1, v2, ..., v5]` potentially improving their sum's bound, especially when the noisy values come from the Laplace mechanism. We leave the integration of other optimizations as future work.

With this generalization DPella is also extensible to other notions of privacy (e.g., Rényi-DP [43]) or other mechanisms by simply declaring a new data type, its principles of composition, a way to sample noise (to compute `noiseScale`) and determine its iCDF (to compute `errorDist`). Additionally, if not Chernoff bound is provided DPella will use union bound instead. All extensions to our framework are delimited and clearly identified thanks to our typed approach.

A.9 Limitations & Extensions

We have discussed so far the use of DPella as an API allowing a programmer to implement her own data analyses. However, we foreseen DPella also serving as a "glue" which enables a programmer to integrate arbitrary DP-algorithms, as (black-box) building blocks while reasoning about accuracy. In this light, our design supports the introduction of new primitives when some analyses cannot be directly implemented because either (i) the static analysis for accuracy provided by DPella is too conservative, or (ii) DPella's API building blocks are not enough to express the desired analysis. Below, we describe several possible extensions.

The matrix mechanism (MM)

As we discussed in the previous section, in some situations DPella allows to answer in an accurate way multiple counting queries in a way that is similar to the MM. As an example, DPella estimates accuracy better than MM for the strategy **I**—recall Section A.6. However, for other workloads and other strategies the accuracy provided by DPella is too conservative. To consider other workloads and strategies, the MM can be incorporated into DPella as a primitive for answering counting queries. The requirements for this are that the return values are tainted, and that we have an iCDFs for it—this can be calculated as in [25]. In general, it is sound to add new primitives which permit a more precise accuracy analysis as long as the return values are tainted, and an accuracy information is provided—thus effectively allowing to further compose the primitive with other analyses by means of the union bound.

Branching on noisy values

By construction, DPella programs are not allowed to branch on results produced by aggregations; this restriction has been enforced since computing the (α, β) -accuracy of such programs poses a challenge in terms of the complexity of their error estimation. More specifically, determining the accuracy of a program branching on a noisy value involves the computation of conditional probabilities (together with the notion of conditional independence); we have identified two main difficulties carried by the consideration of this measurement. Firstly, it must account for both branches' error, thus, quickly loosening the bounds as the program's complexity increases. Secondly, it is challenging to define a general and compositional way of reasoning about the accuracy of the combination of such programs and their independence tracking. To overcome this limitation, we have proposed adding programs that rely on branching over noisy values (e.g., SVT) as black-box primitives in DPella; below, we elaborate on this approach.

Primitives with non-compositional privacy analyses

Several DP-algorithms have a privacy analysis which does not follow directly by composition. Some well-known examples are report-noisy-max, the exponential mechanism, and the sparse-vector technique—see [19] and [9] for more details. In their natural implementations, these algorithms branch on the result of some noised query's result, and the privacy analyses use some properties of the noise distributions that are not directly expressible in terms of composition of differentially private components. Because DPella's API does not allow to branch on the results of noised queries, and because the privacy analyses that DPella support are based on composition, we cannot implement these analyses directly using the DPella API. However, we can provide them as (black-box) primitives. We already discussed how to integrate report-noisy-max through a primitive `dpMax` (Figure A.4). The exponential mechanism (EM) can be incorporated into DPella in a similar way. A subtlety that one has to consider is the fact that the privacy guarantee of EM depends on a bound of the sensitivity of the score function. We handle this by requiring

the score function’s output to be bound between 0 and 1, bounding the sensitivity to be at most 1. As with `dpMax`, the output of EM is tainted. The EM is an important mechanism which allows to implement many other techniques. In particular, we can use EM to implement the offline version of the sparse vector technique, as discussed in [19]. These components allow DPella to support automated reasoning about accuracy for complex algorithms such as the offline version of the MWEM algorithm [27] following an analysis similar to the one discussed in [8].

Generally, one of the biggest challenges of introducing black-box primitives into DPella is to determine their (α, β) -accuracy. There is a significant disparity in how the accuracy of well-known differentially private routines is determined in the literature; thus, there is no straightforward approach to translate such results into our error measurement—if possible at all. Recent work from Barthe *et al.* [3] provides a uniform definition of accuracy across differentially-private routines; we consider these results a promising starting point to systematically interpret these algorithms’ accuracy calculations into DPella’s error measurement.

Online adaptive algorithms

Several DP-algorithms have different implementations depending if they work offline — where all the decisions are taken upfront before running the program — or online — where some of the decisions are taken while running the program. Online algorithms usually have a more involved control flow which depends on information that are available at runtime. As an example, the online version of the sparse vector technique uses the result of a DP query to decide whether to stop or not the computation (or whether to stop or not giving meaningful answers). These kind of algorithms usually are based on some re-use of a noised result which correspond to a taint value in DPella. So, the current design of DPella cannot support them. We plan to explore as future how to integrate these algorithms in DPella.

Improving accuracy through post-processing

Several works have explored the use of post-processing techniques to improve on accuracy, e.g. [30, 28, 48]. Most of these works use accuracy measure that differ from the one we consider here, and use some specific properties of the particular problem at hand. As an example, the work by Hay *et al.* [30] describes how to boost accuracy in terms of *Mean Squared Error* (MSE) for DP hierarchical queries by post-processing the DP results by means of some relatively simple optimization. This improvement in accuracy relies among other things on the impact that the optimization has on the MSE, which does not directly apply to the α - β notion of accuracy we use here. We expect that, also for the notion of α - β accuracy we use, it is possible to use post-processing for improve accuracy. However, we leave this for future works. Moreover, the reason for us to chose α - β accuracy as the principal notion of accuracy in DPella is because of its compositional nature expressible through the use of probability bounds. It is an interesting future direction to design a similar compositional theory also for other accuracy notions such as MSE. We expect DPella to be extensible to incorporate such a theory, once it is available.

A.10 Related work

Programming frameworks for DP

PINQ [28] uses dynamic tracking and sensitivity information to guarantee privacy of computations. Among the frameworks and tools sharing features with PINQ we highlight: Airavat [52]; wPINQ [50]; DJoin [46]; Ektelo [45]; Flex [32]; and PrivateSQL [33]. In contrast to DPella, none of these works keeps track of accuracy, nor static analysis for privacy or accuracy. As discussed in Section A.4, DPella supports a limited form of joins, and it is still able to provide accuracy estimates. We leave as future work to support more general join operations through techniques similar to the ones proposed in Flex and PrivateSQL. While several of the components from the frameworks discussed above are not supported in the current implementation of DPella, these can be added as black-box primitives, as we discussed in Section A.9. All the programming frameworks discussed above support reasoning about privacy for complex data analyses while neglecting accuracy, whereas DPella supports accuracy, but restricts the programming framework to rule out certain analysis (e.g., adaptive ones) for which we do not have a general compositional theory, yet.

Tools for DP

In a way similar to DPella, there exist tools which support reasoning about accuracy and restrict the kind of data analyses they support. GUPT [45] is a tool based on the sample-and-aggregate framework for differential privacy [49]. GUPT allows analysts to specify the target accuracy of the output, and compute privacy from it—or vice versa. This approach has inspired several of the subsequent works and also our design. The limitations of GUPT are that it supports only analyses that fit in the sample-and-aggregate framework, and it supports only confidence intervals estimates expressed at the level of individual queries. In contrast, DPella supports analyses of a more general class, such as the ones we discussed in Section A.3 and Section A.6, and it also allows to reason about the accuracy of combined queries, rather than just about the individual ones. PSI [24] offers to the data analyst an interface for selecting either the level of accuracy that she wants to reach, or the level of privacy she wants to impose. The error estimates that PSI provides are similar to the ones that are supported in DPella. However, similarly to GUPT, PSI supports only a limited set of transformations and primitives, it supports only confidence intervals at the level of individual queries, and in its current form it does not allow analysts to submit their own (programmed) queries.

APEx [25] has similar goals as DPella and it allows data analysts to write queries as SQL-like statements. However, the model that APEx uses is different from DPella's. It supports three kind of queries: WCQ (counting queries), ICQ (iceberg counting queries), and TCQ (top-k counting queries). To answer WCQ queries, APEx uses the matrix mechanism (recall Section A.6) and applies a Monte Carlo simulations to achieve accuracy bounds in terms of α and β , and to determine the least privacy parameter (ϵ) that fits those bounds. We have shown how DPella can be used to answer queries based on the identity strategies using partition and concentra-

tion bounds. To answer effectively different workloads and strategies as well as ICQ and TCQ queries, we would need to extend DPella with the matrix mechanism as a black-box (recall Section A.9). While APEx supports advanced query strategies, it does not provide means to reason about combinations of analyses, e.g., it does not support reasoning about the accuracy of a query using results from WCQs queries to perform TCQs ones. DPella instead has been designed specifically to support the combination of different queries. As we discussed in Section A.9, DPella can be seen as a programming environment that could be combined with some of the analyses supported by tools similar to PSI, GUPT or APEx in order to reason about the accuracy of the combined queries.

Formal Calculi for DP

There are several works on enforcing differential privacy relying on different models and techniques. Within this group are Fuzz [33]—a programming language which enforces (pure) differential privacy of computations using a linear type system which keeps track of program sensitivity—and its derivatives DFuzz [22], Adaptive Fuzz [43], Fuzzi [65], and Duet [47]. Hoare2 [7], a programming language which enforces (pure or approximate) differential privacy using program verification, together with its extension PrivInfer [6] supporting differentially private Bayesian programming; and other systems using similar ideas [9, 1, 44, 42].

Barthe et al. [6] devise a method for proving differential privacy using Hoare logic. Their method uses accuracy bounds for the Laplace Mechanism for proving privacy bounds of the Propose-Test-Release Mechanism, but cannot be used to prove accuracy bounds of arbitrary computations. Later, Barthe et al. [8] develop a Hoare-style logic, named aHL, internalizing the use of the union bound for reasoning about probabilistic imperative programs. The authors show how to use aHL for reasoning in a mechanized way about accuracy bounds of several basic techniques such as report-noisy-max, sparse vector and MWEM. This work has largely inspired our design of DPella but with several differences. First, aHL mixes the reasoning about accuracy with the more classical Hoare-style reasoning. This choice makes aHL very expressive but difficult to automate. DPella instead favors automation over expressivity. As discussed before, the use of DPella to derive accuracy bound is transparent to a programmer thanks to its automation. On the other hand, there are mechanisms that can be analyzed using aHL and cannot be analyzed using DPella, e.g. adaptive online algorithms. Second, aHL supports only reasoning about accuracy but it does not support reasoning about privacy. This makes it difficult to use aHL for reasoning about the privacy-accuracy trade-offs. Finally, aHL supports only reasoning using the union bound and it does not support reasoning based on the Chernoff bound. This makes DPella more precise on the algorithms that can be analyzed using the Chernoff Bound. Barthe et al [5] use aHL, in combination with a logic supporting reasoning by coupling, to verify differentially private algorithms whose privacy guarantee depends on the accuracy guarantee of some sub-component. We leave exploring this direction for future work. More recently, Smith et al. [57] propose an automated approach for computing accuracy bounds of probabilistic imperative programs. This work shares some similarities with our. However, it does not

support reasoning about privacy, and it only uses the Union Bound and do not attempt to reason about probabilistic independence to obtain tighter bounds.

Other works

In a recent work, Ligett et al. [37] propose a framework for developing differentially private algorithms under accuracy constraints. This allows one to chose a given level of accuracy first, and then finding the private algorithm meeting this accuracy. This framework is so far limited to empirical risk minimization problems and it is not supported by a system, yet.

A.11 Conclusions

DPella is a programming framework for reasoning about privacy, accuracy, and their trade-offs. DPella uses taint analysis to detect probabilistic independence and derive tighter accuracy bounds using Chernoff bounds. We believe the principles behind DPella, i.e., the use of concentration bounds guided by taint analysis, could generalize for more notions of privacy such as Rényi-DP [43], concentrated differential privacy [18], zero concentrated differential privacy [12], or truncated concentrated differential privacy [11] (as done with (ϵ, δ) -DP). As future work, we envision lifting the restriction that programs should not branch on query outputs.

Acknowledgments

We thank the anonymous reviewers for constructive feedback on an earlier version of this work. We would like to thank Gilles Barthe for early feedback on the development of DPella. This work was initiated by a STINT Initiation grant (IB 2017-77023) and supported by the Swedish Foundation for Strategic Research (SSF) under the project Octopi (Ref. RIT17-0023) and WebSec (Ref. RIT17-0011) as well as the Swedish research agency Vetenskapsrådet. Marco Gaboardi's work was partially funded by the National Science Foundation under Grants No. 1718220 and 1845803.

Bibliography

- [1] A. Albarghouthi and J. Hsu. Synthesizing coupling proofs of differential privacy. *PACMPL*, 2(POPL), 2018.
- [2] B. Balle and Y.-X. Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. *arXiv preprint arXiv:1805.06530*, 2018.
- [3] G. Barthe, R. Chadha, P. Krogmeier, A. P. Sistla, and M. Viswanathan. Deciding accuracy of differential privacy schemes. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–30, 2021.
- [4] G. Barthe, G. P. Farina, M. Gaboardi, E. J. G. Arias, A. Gordon, J. Hsu, and P. Strub. Differentially private bayesian programming. In *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2016.
- [5] G. Barthe, N. Fong, M. Gaboardi, B. Grégoire, J. Hsu, and P. Strub. Advanced probabilistic couplings for differential privacy. In *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2016.
- [6] G. Barthe, M. Gaboardi, E. J. G. Arias, J. Hsu, C. Kunz, and P. Strub. Proving differential privacy in Hoare logic. In *Proc. IEEE Computer Security Foundations Symposium*, 2014.
- [7] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, A. Roth, and P.-Y. Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. In *POPL '15*. ACM, 2015.
- [8] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P. Strub. A program logic for union bounds. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 55 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- [9] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P. Strub. Proving differential privacy via probabilistic couplings. In *Proc. ACM/IEEE Symposium on Logic in Computer Science*, 2016.
- [10] J. Blocki, A. Blum, A. Datta, and O. Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Innovations in Theoretical Computer Science, ITCS*, 2013.
- [11] M. Bun, C. Dwork, G. N. Rothblum, and T. Steinke. Composable and versatile privacy via truncated cdp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 74–86. ACM, 2018.
- [12] M. Bun and T. Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*. Springer, 2016.

- [13] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Transactions on Information and System Security (TISSEC)*, 14(3):26, 2011.
- [14] G. Cormode, T. Kulkarni, and D. Srivastava. Marginal release under local differential privacy. In *Proc. of International Conference on Management of Data, SIGMOD*, pages 131–146, 2018.
- [15] D. P. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
- [16] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography, TCC’06*, pages 265–284, 2006.
- [17] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
- [18] C. Dwork and G. N. Rothblum. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887*, 2016.
- [19] C. Dwork, G. N. Rothblum, and S. P. Vadhan. Boosting and differential privacy. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 51–60, 2010.
- [20] H. Ebadi and D. Sands. Featherweight PINQ. *Privacy and Confidentiality*, 7(2), 2017.
- [21] R. A. Eisenberg, D. Vytiniotis, S. L. Peyton Jones, and S. Weirich. Closed type families with overlapping equations. In *The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2014.
- [22] M. Gaboardi, E. J. G. Arias, J. Hsu, A. Roth, and Z. S. Wu. Dual query: Practical private query release for high dimensional data. In *Proc. International Conference on Machine Learning, ICML*, 2014.
- [23] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2013.
- [24] M. Gaboardi, J. Honaker, G. King, K. Nissim, J. Ullman, and S. P. Vadhan. PSI (Ψ): a private data sharing interface. *CoRR*, abs/1609.04340, 2016.
- [25] C. Ge, X. He, I. F. Ilyas, and A. Machanavajjhala. APEX: Accuracy-aware differentially private data exploration. In *Proc. International Conference on Management of Data*, 2019.
- [26] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *Proc. of USENIX Security Symposium*, 2011.

- [27] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*, 2012.
- [28] M. Hardt and K. Talwar. On the geometry of differential privacy. In *Proc. of the 42nd ACM Symposium on Theory of Computing, STOC*, 2010.
- [29] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using DPBench. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, 2016.
- [30] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1), 2010.
- [31] N. M. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for SQL queries. *PVLDB*, 11(5), 2018.
- [32] N. M. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for SQL queries. *PVLDB*, 11(5), 2018.
- [33] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. PrivateSQL: A differentially private SQL query engine. *Proc. VLDB Endow.*, 12(11):1371–1384, July 2019.
- [34] C. H. Lampert, L. Ralaivola, and A. Zimin. Dependency-dependent bounds for sums of dependent random variables. *arXiv preprint arXiv:1811.01404*, 2018.
- [35] C. Li, G. Miklau, M. Hay, A. McGregor, and V. Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *VLDB J.*, 24(6), 2015.
- [36] P. Li and S. Zdancewic. Arrows for secure information flow. *Theoretical Computer Science*, 411(19):1974–1994, 2010.
- [37] K. Ligett, S. Neel, A. Roth, B. Waggoner, and Z. S. Wu. Accuracy first: Selecting a differential privacy level for accuracy-constrained ERM. *CoRR*, abs/1705.10829, 2017.
- [38] E. Lobo-Vesga, A. Russo, and M. Gaboardi. A programming framework for differential privacy with accuracy concentration bounds. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1333–1350, Los Alamitos, CA, USA, may 2020. IEEE Computer Society.
- [39] A. Machanavajjhala, D. Kifer, J. M. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *Proc. International Conference on Data Engineering, ICDE*, 2008.
- [40] F. McSherry and R. Mahajan. Differentially-private network trace analysis. *ACM SIGCOMM Computer Communication Review*, 41(4):123–134, 2011.

- [41] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*. ACM, 2009.
- [42] D. J. Mir, S. Isaacman, R. Cáceres, M. Martonosi, and R. N. Wright. DP-WHERE: differentially private modeling of human mobility. In *Proc. IEEE International Conference on Big Data*, 2013.
- [43] I. Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 2017.
- [44] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.
- [45] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. E. Culler. GUPT: privacy preserving data analysis made easy. In *Proc. ACM SIGMOD International Conference on Management of Data, SIGMOD*, 2012.
- [46] A. Narayan and A. Haeberlen. DJoin: Differentially private join queries over distributed databases. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI*. USENIX Association, 2012.
- [47] J. P. Near, D. Darais, C. Abuah, T. Stevens, P. Gaddamadugu, L. Wang, N. Soman, M. Zhang, N. Sharma, A. Shan, and D. Song. Duet: An expressive higher-order language and linear type system for statically enforcing differential privacy. *Proc. ACM Program. Lang.*, 3(OOPSLA), Oct. 2019.
- [48] A. Nikolov, K. Talwar, and L. Zhang. The geometry of differential privacy: the sparse and approximate cases. In *Symposium on Theory of Computing Conference, STOC’13*, 2013.
- [49] K. Nissim, S. Raskhodnikova, and A. D. Smith. Smooth sensitivity and sampling in private data analysis. In *Proc. Annual ACM Symposium on Theory of Computing*, 2007.
- [50] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis. *PVLDB*, 7(8), 2014.
- [51] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *Proc. ACM SIGPLAN International Conference on Functional Programming*, 2010.
- [52] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for MapReduce. In *Proc. USENIX Symposium on Networked Systems Design and Implementation, NSDI*, 2010.
- [53] A. Russo. Functional Pearl: Two Can Keep a Secret, if One of Them Uses Haskell. In *Proc. of the ACM SIGPLAN International Conference on Functional Programming*. ACM, 2015.
- [54] A. Russo, K. Claessen, and J. Hughes. A library for light-weight information-flow security in Haskell. In *Proc. ACM SIGPLAN Symp. on Haskell*. ACM Press, 2008.

- [55] A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, Jan. 2003.
- [56] D. Schoepe, M. Balliu, B. C. Pierce, and A. Sabelfeld. Explicit secrecy: A policy for taint tracking. In *IEEE European Symposium on Security and Privacy*, pages 15–30, 2016.
- [57] C. Smith, J. Hsu, and A. Albarghouthi. Trace abstraction modulo probability. *PACMPL*, 3(POPL), 2019.
- [58] D. Terei, S. Marlow, S. L. Peyton Jones, and D. Mazières. Safe Haskell. In *Proceedings of the 5th ACM SIGPLAN Symposium on Haskell, Haskell 2012, Copenhagen, Denmark, 13 September 2012*, pages 137–148, 2012.
- [59] J. Thaler, J. Ullman, and S. P. Vadhan. Faster algorithms for privately releasing marginals. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP*, pages 810–821, 2012.
- [60] Y. Wang, Z. Ding, G. Wang, D. Kifer, and D. Zhang. Proving differential privacy with shadow execution. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019.
- [61] D. Winograd-Cort, A. Haeberlen, A. Roth, and B. C. Pierce. A framework for adaptive differential privacy. *PACMPL*, 1(ICFP), 2017.
- [62] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Trans. Knowl. Data Eng.*, 23(8), 2011.
- [63] D. Zhang and D. Kifer. LightDP: towards automating differential privacy proofs. In *Proc. ACM SIGPLAN Symp. on Principles of Programming Languages*, 2017.
- [64] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. EKTELO: A framework for defining differentially-private computations. In *Proc. International Conference on Management of Data*, 2018.
- [65] H. Zhang, E. Roth, A. Haeberlen, B. C. Pierce, and A. Roth. Fuzzi: A three-level logic for differential privacy. In *Proc. ACM SIGPLAN International Conference on Functional Programming (ICFP’19)*, 2019.



Sensitivity by Parametricity: Simple Sensitivity Proofs for Differential Privacy

Abstract. The work of Fuzz has pioneered the use of functional programming languages where types allow reasoning about the sensitivity of programs. Fuzz and subsequent work (e.g., DFuzz and Duet) use advanced technical devices like linear types, modal types, and partial evaluation. These features usually require the design of a new programming language from scratch—a major task on its own! While these features are part of the classical toolbox of programming languages, they might be unfamiliar to non-programming language experts. In this work, we propose to take a different direction. We present the novel idea of applying *parametricity*, i.e., a well-known abstract uniformity property enjoyed by polymorphic functions, to compute sensitivity of functions. A direct consequence of our result is that calculating the sensitivity of functions can be reduced to simply type-checking in a programming language with support for polymorphism. We formalize our main result in a calculus, prove its soundness, and implement a software library in the programming language Haskell—where we reason about the sensitivity of canonical examples. We also show that thanks to type-inference, our approach supports a limited form of sensitivity inference—something that, to the best of our knowledge, has not been explored before. Our library, called SPAR, is implemented in 365 lines of code.

B.1 Introduction

Differential privacy (DP) is a mathematical definition of privacy that challenges the paradox of learning helpful information about a population while protecting the data of individuals. The standard way to achieve DP is by computing the desired analysis in a dataset and adding calibrated noise before publication [18]. This simple idea has spawned a series of works (e.g., [28, 45, 27, 29, 2]) on designing programming languages to help analysts implement differentially-private analyses accessing sensitive information. At the backbone of every DP programming language resides the noise-calibration mechanism, which determines how much noise is required to mask an individual’s membership to the population depending on the query’s global sensitivity—how volatile a function can be concerning changes in its inputs.

Unfortunately, automatically calculating the global sensitivity of arbitrary functions can be challenging. Current DP programming languages provide data analysts with a limited set of pre-defined functions whose global sensitivity is known a priori. However, even though the use of pre-defined functions has allowed for many exciting analyses, it severely constrains the kind of queries one can express. Aiming to tackle this issue, Reed and Pierce [33] designed Fuzz, a functional programming language that uses linear indexed types (with indexes representing sensitivities) to track every expressible program’s sensitivity. Their approach has been further developed in a series of works [22, 20, 43, 29], incorporating additional programming language features such as partial evaluation, linear, and modal types, thus extending Fuzz’s expressivity. However, these features are not mainstream and usually require the design of a new language from scratch—a significant task that can significantly hinder the adoption of those ideas. Moreover, features like linear and modal types are rarely known outside academic circles, which also constitute a significant barrier of entry to the non-programming language expert.

In a concurrent work, Abuah *et al.* [2] introduced SOLO, a type system for differential privacy where programs’ sensitivity is determined with respect to a set of data sources instead of their variables; thus avoiding linear typing disciplines. In their approach, base types are annotated with sensitivity information tracked and altered by the various operations in a taint-analysis fashion. The soundness of SOLO’s sensitivity analysis is then proven via metric preservation. While SOLO showcases that features available on mainstream richly-typed programming languages are sufficient to compute the sensitivity of user-defined functions statically, their system does not allow users to implement higher-order recursive functions (such as `map`). To overcome this limitation, higher-order functions are added as trusted primitives and their soundness is left to be proven by the authors.

In this work, we explore a different direction. We propose a novel use of *parametricity*—a well-known abstract uniformity property enjoyed by *polymorphic functions*—that in combination with type constraints [40, 26] and type-level numbers can verify the sensitivity of functions—including higher-order ones—by simply type-checking.

We formalize our ideas in λ_{SPAR} : a sound calculus capable of proving the sensitivity of programs by just relying on features currently available in strongly-typed declarative programming languages. We also present SPAR, a concrete implementation of λ_{SPAR} as a library for the Haskell programming language. We showcase how SPAR can be used to reason about the sensitivity of classical examples like summing, mapping, and sorting elements of a vector. Because SPAR is provided as an embedded domain-specific language, our implementation leverages Haskell’s advanced type-inference to provide limited support for sensitivity inference—a feature that, to the best of our knowledge, has not been explored before. We argue that the main result of this work opens the door to integrating procedures for automatically proving the sensitivity of functions into the programming workflow, e.g., by using SPAR’s sensitivity proofs as an input to other Haskell-based DP frameworks [27].

In summary, the main contributions of this paper are as follows:

- a novel use of parametrically polymorphic functions to prove sensitivity of functions,
- a formal calculus that captures our ideas (Section B.2), and a soundness proof of our claims (Section B.3.1),
- an implementation of our calculus as a library in Haskell with case studies (Sections B.4), and
- an analysis about how to realize our calculus in another programming languages (Section B.5).

B.1.1 Motivating examples

Before we dive into our formalism and soundness proof, we showcase our main contributions by examples in our software library SPAR. The library is thought for developers to write functions and, by doing so, *discover* and *provide a proof* of their sensitivity. Despite SPAR being implemented in the functional programming language Haskell, we argue that the ideas presented here can be deployed in programming languages supporting parametric polymorphism and a type system with support for type-level natural numbers—a detailed discussion about the required features in Section B.5.

We consider a function to be k -sensitive (or have sensitivity k) if it magnifies the distance of its inputs by a factor of *at most* k . Formally:

Definition B.1 (Sensitivity). *Given two metric spaces (A, d_A) and (B, d_B) , a function $f : A \rightarrow B$ is k -sensitive iff:*

$$\begin{aligned} \forall n \in \mathbb{R}, (x_1, x_2) \in A. \\ d_A(x_1, x_2) \leq n \Rightarrow d_B(f(x_1), f(x_2)) \leq k * n \end{aligned}$$

We start by considering a simple function that adds the constant 42 to any given numerical value; in Haskell, we write:

```
add42 :: Int → Int
add42 x = x+42
```

If d_{Int} (for brevity d) is defined as the euclidean distance, we can intuitively notice that `add42` is 1-sensitive since for any two possible inputs at a certain distance, it produces outputs that are at the same distance. For instance, if we consider inputs 5 and 23, where $d(5, 23) = 18$, then outputs 47 and 65 are also at distance 18, i.e., $d(47, 65) = 18$.

User-defined functions, however, are often more complex than `add42`, and as functions' complexity increases, the less intuitive it's to reason about their sensitivity. For instance, the following function utilizes its input several times to create a series of nested pairs:

```
nest :: Int → (Int, (Int, (Int, Int)))
nest x = (x, (add42 x, (x, x)))
```

What is `nest`'s sensitivity? While the sensitivity-knowledgeable reader could quickly answer this question, the everyday programmer might struggle to perform such a cumbersome analysis. More importantly, we argue that the responsibility of conducting such critical calculations should not fall on the (error-prone) programmer but rather on its (reliable) programming tools.

With this in mind SPAR introduces the following abstract datatype to write programs:

```
data Rel (d :: Nat) a
```

This data type is indexed by type-level natural numbers ($d :: \text{Nat}$), where d stands for distance. When programming, developers can think about the term $t :: \text{Rel } d \ a$ simply as a term of type a —i.e., $\text{Rel } d \ a$ is isomorphic to a . However, for sensitivity calculations, a term $t :: \text{Rel } d \ a$ will be interpreted as the *collection* of all pairs of values of type a whose distance is at most d . From now on, we will refer to the terms $t :: \text{Rel } d \ a$, for a given d and a , as relational terms or values.

Together with the data type $\text{Rel } d \ a$, SPAR exposes a set of basic relational operations such as:

```
lit  :: Int → Rel d Int
(:+:) :: Rel d1 Int → Rel d2 Int → Rel (d1+d2) Int
(:*)  :: Rel da a → Rel db b → Rel (da+db) (a, b)
```

In essence, the types of these primitives encode how *distances of the resulting relational values change with respect to the distances of the inputs*. Primitive `lit` x lifts a regular integer into a relation one, which means that an integer can be at any distance from another one—thus, the distance is parametric on d . Primitive `(:+:)` indicates that the distance of added values is, at most, $d_1 + d_2$. The primitive `(:*)` creates relational pairs at distance $d_a + d_b$, thus encoding pairs under the ℓ_1 norm. For simplicity, we demonstrate how SPAR works for the norm ℓ_1 but other norms like ℓ_∞ are possible.

With these simple operations, we can rewrite our previous examples as SPAR functions:

```
add42 x = x :+ (lit @ 0 42)
nest  x = x :* (add42 x :* (x :* x))
```

where we use Haskell's type applications (`@`) to indicate that `42` will be a constant, thus its distance is set to 0. Aided by Haskell's type-system we can inspect how much `add42` and `nest` magnify the distance between their inputs.

```
> : type add42
Rel d Int → Rel d Int
> : type nest
Rel d Int → Rel (4*d) (Int, (Int, (Int, Int)))
```

Function `add42` preserves its inputs distances while `nest` quadruples them. SPAR, by construction, tracks how many occurrences of x affect the distance inferred for

the results—which diverges from previous work [33, 22, 29] in requiring the utilization of linear type-systems. Moreover, since the code that we wrote is generic, the type signatures produced by the type-system are *polymorphic* in d . In other words, the code of `add42` and `nest` can be applied to relational terms at distance 1 (i.e., terms of type $\text{Rel } 1 \ a$), terms at distance 2 (i.e., terms of type $\text{Rel } 2 \ a$), etc.

The central insight of this work is that parametric polymorphism can capture the fact that outputs’ distances in functions are bounded in the same manner independently of the inputs’ distances, which is no more than the definition of sensitivity! For instance, the type of `nest` indicates that this function magnifies the distance of its inputs by a factor of 4 , thus `nest` has sensitivity 4 .

SPAR uses the type synonym

```
type Sen (k :: Nat) a b =  $\forall d . \text{Rel } d \ a \rightarrow \text{Rel } (k*d) \ b$ 
```

to directly refer to functions from a to b with proven sensitivity k . The proof comes from the explicit use of parametricity. Specifically, to create a function of type $\text{Sen } k \ a \ b$, we need to implement a *polymorphic function on the distance of its inputs* (observe the $\forall d$ in the type-signature) whose outputs’ distance is scaled by k .

With this data type, we can provide a proof of sensitivity for our previous examples:

```
sen_add42 :: Sen 1 Int Int
sen_add42 = add42

sen_nest :: Sen 4 Int (Int, (Int, (Int, Int)))
sen_nest = nest
```

Observe that the definitions of `sen_add42` and `sen_nest` do not require us to perform any transformations to our functions, meaning that the proof of sensitive consists in just being able to type-check these expressions.

Parametric polymorphism gives us a simple proof mechanism for sensitivity, but how far can we go with it? How expressive our programs can be? In the following sections, we show that the core calculus of this library is sound and how it can cover some advanced examples similar to the ones found in previous work [33, 22, 29].

B.2 λ_{SPAR} : a calculus for distance tracking

In this section, we present λ_{SPAR} , a calculus that annotates types with *distances* and keep track of them using special operators. Importantly, functions in the calculus can be defined on parametric distances, this feature is key to obtaining proof of their sensitivity. For clarity, this section will introduce a simplified version of λ_{SPAR} showcasing the main technical ideas; however, we will let the reader know when these simplifications occur.

B.2.1 Syntax

Types The syntax of types is described by the following grammar:

$$i \in \text{dvar} \quad r \in \mathbb{R}_{\geq 0} \quad l \in \mathbb{N}$$

$$\begin{aligned}
\tau \in \text{type} & ::= \sigma \mid \tau \times \tau \mid \vec{\tau}_l \mid \tau \rightarrow \tau \mid \text{Rel } d \sigma \\
\sigma \in \text{type} & ::= \mathbb{R} \mid \sigma \times \sigma \mid \vec{\sigma}_l \\
d \in \text{dist} & ::= i \mid r \mid d + d \mid d * d
\end{aligned}$$

Our formalism includes a basic numeric type \mathbb{R} , pairs $(_ \times _)$, functions $(_ \rightarrow _)$, fixed-length lists or vectors $(\vec{_})_l$, and the novel relational type $\text{Rel } d (_)$. As previously stated, the relational type is essentially a regular type annotated with an upper bound on the distance between its inhabitants. To keep distance-reasoning as intuitive and straightforward as possible, we introduce a hierarchical structure in the types preventing nesting within relational types. The reader should consider the overloaded usage of $(_ \times _)$ and $(\vec{_})_l$ as one and the same.

Distances are represented as terms in a small language at the type level including variables i , non-negative real constants r (i.e., $r \in \mathbb{R}_{\geq 0}$), and two operators denoting addition $(_ + _)$ and multiplication $(_ * _)$. We remark that, without loss of generality, the implementation of λ_{SPAR} in Section B.4 encodes distances as type-level natural numbers with their respective operations. Vectors' lengths are represented by type-level natural numbers. The opaque expression $l \in \mathbb{N}$ captures the fact that there is a small language for natural numbers with variables similar to that of distances and DFuzz's size terms. In favor of readability, we omit the details of such a language.

Contrary to previous calculus for sensitivity analysis, λ_{SPAR} 's types do not carry *sensitivity* information associated with variables (as in Fuzz [33]) or values (as in SOLO [2]). Instead, the use of relational types to track *distances* allow us to explicitly model bounded metric spaces from which function sensitivity can be *proven* as a uniform continuity property.

Terms The grammar for expressions is straightforward. It includes the canonical introduction and elimination forms of an ordinary typed functional programming language: variables, literals and arithmetic operations, pairs, and projections, abstractions and applications, and lists. Moreover, for each non-relational term—excluding variables—there exists its relational counterpart marked with a distinguishing **color**. The only difference between non-relational and relational terms resides in how pairs get eliminated. While non-relational pairs use projections (terms **fst** and **snd**), the relational ones are eliminated via pattern-matching (term **let** $(x, y) = e$ **in** e).

$$x \in \text{evar} \quad n \in \mathbb{R}$$

$$\begin{aligned}
e \in \text{expr} & ::= x \mid n \mid e + e \mid (e, e) \mid \mathbf{fst} \, e \mid \mathbf{snd} \, e \\
& \quad \mid \lambda x. e \mid e @ e \\
& \quad \mid [] \mid e :: e \mid \mathbf{case} \, e \, \mathbf{of} \, \{[] . e\} \{ (x :: xs) . e \} \\
& \quad \mid \mathbf{N} \, e \mid e + e \mid (e, e) \mid \mathbf{let} \, (x, y) = e \, \mathbf{in} \, e \\
& \quad \mid [] \mid e :: e \\
& \quad \mid \mathbf{case} \, e \, \mathbf{of} \, \{[] . e\} \{ (x :: xs) . e \}
\end{aligned}$$

$$\begin{array}{c}
\text{T.NUM}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e : \mathbb{R} \quad \text{vars}(d) \subseteq \Psi}{\Psi; \mathcal{C}; \Gamma \vdash \mathbf{N} e : \text{Rel } d \mathbb{R}} \\
\\
\text{T.ADD}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \mathbb{R} \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \mathbb{R} \quad i \notin \mathbf{FV}(\Psi)}{\Psi, i; \mathcal{C} \wedge i = (d_1 + d_2); \Gamma \vdash e_1 + e_2 : \text{Rel } i \mathbb{R}} \\
\\
\text{T.PAIR}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \sigma_1 \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \sigma_2 \quad i \notin \mathbf{FV}(\Psi)}{\Psi, i; \mathcal{C} \wedge i = (d_1 + d_2); \Gamma \vdash (e_1, e_2) : \text{Rel } i (\sigma_1 \times \sigma_2)} \\
\\
\text{T.}\sqsubseteq \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e : \tau_1 \quad \Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_2}{\Psi; \mathcal{C}; \Gamma \vdash e : \tau_2} \\
\\
\text{T.LET}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d (\sigma_1 \times \sigma_2) \quad \Psi, i_1, i_2; \mathcal{C} \wedge d = (i_1 + i_2); \Gamma, x : \text{Rel } i_1 \sigma_1, y : \text{Rel } i_2 \sigma_2 \vdash e_2 : \tau \quad i_1, i_2 \notin \mathbf{FV}(\Psi)}{\Psi; \mathcal{C}; \Gamma \vdash \mathbf{let} (x, y) = e_1 \mathbf{in} e_2 : \tau}
\end{array}$$

Figure B.1: Selected typing rules for relational terms

Typing rules The inference rules in Figure B.1 define the typing relation for relational. Type judgments for non-relational expressions are entirely routine. Besides the usual environment Γ for term variables, a judgment $\Psi; \mathcal{C}; \Gamma \vdash e : \tau$ contains two extra parameters, Ψ and \mathcal{C} , conforming to the grammar:

$$\begin{array}{ll}
\Gamma \in \text{tenv} & ::= \emptyset \mid \Gamma, x : \tau \\
\Psi \in \text{denv} & ::= \emptyset \mid \Psi, i \\
\mathcal{C} \in \text{cenv} & ::= \emptyset \mid d = d \mid \mathcal{C} \wedge \mathcal{C}
\end{array}$$

Set Ψ represents the environment for distance variables. Distinctly from Γ , this environment does not map variables to a type (or *kind*) since all distances are implicitly typed as positive real numbers. The practical effect of allowing variables within distance expressions is that it models the effect of *polymorphism* on distances. Evidently, a term $\Psi, i; \mathcal{C}; \Gamma \vdash e : \text{Rel } i \mathbb{R}$ can be interpreted as a term of type $\text{Rel } 5 \mathbb{R}$ or $\text{Rel } 1 \mathbb{R}$ under the substitutions $[i := 5]$ and $[i := 1]$, respectively. In Haskell, we write $e :: \forall i . \text{Rel } i \text{ Int}$ and apply the substitutions with $e@5$ and $e@1$. This insight will be explored in Section B.4.

Set \mathcal{C} records the constraints under which typing is obtained. These constraints are carried out to ensure that the interpretation of expressions with free (distance) variables preserve the metric of their respective types. For instance, the consequent

$$\Psi; \mathcal{C} \models d_1 \leq d_2 \iff \forall \rho. \Psi \subseteq \text{dom}(\rho) \wedge \rho \models \mathcal{C} \Rightarrow \llbracket d_1 \rrbracket_\rho \leq \llbracket d_2 \rrbracket_\rho$$

$$\begin{array}{c}
\text{ST.REFL} \\
\hline
\Psi; \mathcal{C} \models \tau \sqsubseteq \tau \\
\\
\text{ST.TRANS} \\
\frac{\Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_2 \quad \Psi; \mathcal{C} \models \tau_2 \sqsubseteq \tau_3}{\Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_3} \\
\\
\text{ST.REL} \\
\frac{\mathcal{C}; \Psi \models d_1 \leq d_2}{\Psi; \mathcal{C} \models \text{Rel } d_1 \sigma \sqsubseteq \text{Rel } d_2 \sigma}
\end{array}$$

Figure B.2: Selected subtyping rules and distance comparison

in rule T.PAIR_R can be read as: no matter which value gets assigned to the distance of the pair (i.e., when substituting variable i) as long as it amounts to the distance of its components—thus, adhering to the ℓ_1 metric.

In a nutshell, the typing rules in Figure B.1 depict how distances are altered depending on the underlying operation. As in Fuzz [33], type safety under this setting not only ensures that “well-typed programs can’t go wrong” but also that “they can’t go too far.” In our case, data type $\text{Rel } d \sigma$ directly encodes how far they can go.

Rules T.NUM_R , T.ADD_R , and T.PAIR_R correspond to the operators `lit`, `(:+:)`, and `(*:)` introduced in Section B.1.1. The rule for relational numbers has the premise $\text{var}(d) \subseteq \Psi$ preventing d from referring to unbounded variables; when a distance expression d satisfies this condition we say that d is *covered by* Ψ . Rule T.LET_R allows deconstructing a relational pair under a scoped environment where the distance of the pair components is represented by the fresh variables i_1 and i_2 , additionally the set of constraints is extended requiring that the addition of these variables is equal to the distance of the original pair; this way, it is ensured that the metric for pairs is preserved under elimination.

Our calculus considers vectors with statically known lengths captured by the type $\vec{\sigma}_l$.

Vector’s introduction and elimination rules are excluded as they require explicit handling of length variables—these details can be found in the accompanying material. Intuitively, given that vectors of fixed lengths can be represented by nested pairs, their introduction, and elimination typing rules impose the same distance constraints as pairs.

Lastly, rule $\text{T.} \sqsubseteq$ follows the standard procedure for subtyping. The main purpose of the subtyping relation \sqsubseteq (see Figure B.2) is to capture the fact that values that are at distance d_1 are also at distance d_2 for $d_1 \leq d_2$. To compare distance expressions under \leq , we provide an interpretation to distance expressions over the domain $\mathbb{R}_{\geq 0}$. Concretely, given an assignment $\rho \in \text{dvar} \rightarrow \mathbb{R}_{\geq 0}$ such that $\text{var}(d) \subseteq \text{dom}(\rho)$, the

inductive interpretation $\llbracket d \rrbracket_\rho$ is defined as:

$$\begin{aligned}\llbracket i \rrbracket_\rho &= \rho(i) \\ \llbracket r \rrbracket_\rho &= r \\ \llbracket d_1 + d_2 \rrbracket_\rho &= \llbracket d_1 \rrbracket_\rho + \llbracket d_2 \rrbracket_\rho \\ \llbracket d_1 * d_2 \rrbracket_\rho &= \llbracket d_1 \rrbracket_\rho * \llbracket d_2 \rrbracket_\rho\end{aligned}$$

Observe that while the arithmetic operators ($+$ and $*$) were symbolic in distance terms, their usage on the right-hand side of the equations refers to the addition and multiplication of real numbers.

With this interpretation, we define an ordering relation over distances $\Psi; \mathcal{C} \models d_1 \dot{\leq} d_2$ taking into account the constraints they must satisfy (see Figure B.2). This relation encapsulates the fact that a term d_1 can be considered less or equal than term d_2 under the constraints \mathcal{C} , only if for every closing assignment ρ satisfying the constraints, the interpretation of d_1 is less or equal than that of d_2 . The subtyping relation relies on such a condition to increase the distance of a relational term (rule ST.REL). In other words, a relational term can only be subtyped if its distance is upgraded (or remains intact).

Lastly, when considering vectors' length variables within the typing rules, sets Ψ and \mathcal{C} are extended to track these variables and their constraints. Accordingly, assignments ρ and interpretations $\llbracket \cdot \rrbracket_\rho$ are adapted to handle distance and length expressions separately.

B.2.2 Operational semantics

We provide a big-step operational semantics with explicit substitutions. The values to which an expression is allowed to evaluate are defined by the following grammar:

$$\begin{aligned}v \in \text{val} \quad & ::= n \mid (v, v) \mid \langle \lambda x. e \mid \gamma \rangle \mid [] \mid v :: v \\ & \mid \mathbf{N} \, v \mid (v, v) \mid [] \mid v :: v\end{aligned}$$

With γ a value environment (also referred to as substitution) mapping variables to values (i.e., $\gamma \in \text{var} \rightarrow \text{val}$). The rules for evaluation are standard (refer to the accompanying material) with judgment $\gamma \vdash e \Downarrow v$ stating that a configuration with value environment γ and term e evaluates to value v .

B.3 Formal guarantees

Soundness for λ_{SPAR} is defined as a *metric preservation* property. Intuitively, the metric preservation property ensures that closing an open term with two distinct but related substitutions will produce related expressions whose distance is bounded.

We adopt the technique of logical relations in which we determine how two well-typed expressions can be considered related at a determined distance. As is

$$\begin{aligned}
\mathcal{D}[\mathbb{R}]_d^\rho &= \{(\mathbf{N} \, n_1, \mathbf{N} \, n_2) \mid |n_1 - n_2| \leq \llbracket d \rrbracket_{\rho, d}\} \\
\mathcal{D}[\sigma_1 \times \sigma_2]_d^\rho &= \{((v_{11}, v_{21}), (v_{12}, v_{22})) \mid \exists d_1, d_2. \rho \models d = d_1 + d_2 \\
&\quad \wedge (v_{11}, v_{12}) \in \mathcal{D}[\sigma_1]_{d_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\sigma_2]_{d_2}^\rho\} \\
\mathcal{D}[\vec{\sigma}_0]_d^\rho &= \{([\], [\])\} \\
\mathcal{D}[\vec{\sigma}_{(l+1)}]_d^\rho &= \{(v_{11} :: v_{21}, v_{12} :: v_{22}) \mid \exists d_1, d_2. \rho \models d = d_1 + d_2 \\
&\quad \wedge (v_{11}, v_{12}) \in \mathcal{D}[\sigma]_{d_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\vec{\sigma}_l]_{d_2}^\rho\} \\
\mathcal{V}[\mathbb{R}]^\rho &= \{(v_1, v_2) \mid v_1 \equiv v_2\} \\
\mathcal{V}[\tau_1 \times \tau_2]^\rho &= \{((v_{11}, v_{21}), (v_{12}, v_{22})) \mid (v_{11}, v_{12}) \in \mathcal{V}[\tau_1]^\rho \wedge (v_{21}, v_{22}) \in \mathcal{V}[\tau_2]^\rho\} \\
\mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho &= \{(\langle \lambda x. e_1 \mid \gamma_1 \rangle, \langle \lambda x. e_2 \mid \gamma_2 \rangle) \mid \exists \Gamma. (\gamma_1, \gamma_2) \in \mathcal{S}[\Gamma]^\rho \\
&\quad \wedge \forall v_1, v_2. (v_1, v_2) \in \mathcal{V}[\tau_1]^\rho \\
&\quad \Rightarrow (\gamma_1[x := v_1] \vdash e_1, \gamma_2[x := v_2] \vdash e_2) \in \mathcal{E}[\tau_2]_{\Gamma, x:\tau_1}^\rho\} \\
\mathcal{V}[\vec{\tau}_0]^\rho &= \{([\], [\])\} \\
\mathcal{V}[\vec{\tau}_{(l+1)}]^\rho &= \{(v_{11} :: v_{21}, v_{12} :: v_{22}) \mid (v_{11}, v_{12}) \in \mathcal{V}[\tau]^\rho \wedge (v_{21}, v_{22}) \in \mathcal{V}[\vec{\tau}_l]^\rho\} \\
\mathcal{V}[\text{Rel } d \, \sigma]^\rho &= \mathcal{D}[\sigma]_d^\rho \\
\mathcal{E}[\tau]_\Gamma^\rho &= \{(\gamma_1 \vdash e_1, \gamma_2 \vdash e_2) \mid \forall \gamma_1, \gamma_2, v_1, v_2. (\gamma_1, \gamma_2) \in \mathcal{S}[\Gamma]^\rho \\
&\quad \wedge \gamma_1 \vdash e_1 \Downarrow v_1 \wedge \gamma_2 \vdash e_2 \Downarrow v_2 \Rightarrow (v_1, v_2) \in \mathcal{V}[\tau]^\rho\} \\
\mathcal{S}[\Gamma]^\rho &= \{(\gamma_1, \gamma_2) \mid \text{dom}(\gamma_1) \equiv \text{dom}(\gamma_2) \equiv \text{dom}(\Gamma) \\
&\quad \wedge \forall (x : \tau). x \in \text{dom}(\Gamma) \Rightarrow (\gamma_1(x), \gamma_2(x)) \in \mathcal{V}[\tau]^\rho\}
\end{aligned}$$

Figure B.3: Mutually-recursive logical relations

customary, we define mutually-recursive logical relations $\mathcal{V}[\![\tau]\!]^\rho$, $\mathcal{E}[\![\tau]\!]_\Gamma^\rho$, and $\mathcal{S}[\![\Gamma]\!]^\rho$ relating values, open expressions and typing contexts, respectively (see Figure B.3). Moreover, we introduce a new relation $\mathcal{D}[\![\sigma]\!]_d^\rho$ specifying the metric relation between relational values (i.e., values of type $\text{Rel } d \sigma$).

All of these relations are indexed with an assignment ρ mapping distance variables to non-negative real numbers. Such assignment is used by the relation $\mathcal{D}[\![\sigma]\!]_d^\rho$ to evaluate the distance term d and use it as an upper bound on the distances of its components. We use $(v_1, v_2) \in \mathcal{D}[\![\sigma]\!]_d^\rho$ to denote that given an assignment ρ , relational values v_1 and v_2 are related at type σ and distance d —similarly for non-relational values, expressions, and environments.

At a high level, our logical relations state:

- Two relational values are related under $\mathcal{D}[\![\sigma]\!]_d^\rho$ if the distance of their operands is less or equal to the value resulting from interpreting the distance term d with the assignment ρ .
- Two values are related under $\mathcal{V}[\![\tau]\!]^\rho$ if they are equivalent (base types) or their components are related accordingly. For example, two functions of type $\tau_1 \rightarrow \tau_2$ are related if they map related inputs $(v_1, v_2) \in \mathcal{V}[\![\tau_1]\!]^\rho$ to related outputs $(\gamma_1 \vdash \lambda x.e_1 @ v_1, \gamma_2 \vdash \lambda x.e_2 @ v_2) \in \mathcal{E}[\![\tau_2]\!]_\Gamma^\rho$.
- Two expressions are related under $\mathcal{E}[\![\tau]\!]_\Gamma^\rho$ when both reduce to values that are related at $\mathcal{V}[\![\tau]\!]^\rho$.
- Two substitutions are related under $\mathcal{S}[\![\Gamma]\!]^\rho$ when both map all of their variables to related values.

Note that these relations depart from previous work as they do not assign a metric interpretation to all of the types in the calculus, instead, such interpretation will be restricted only to relational types. More precisely, those expressions where none of its components have been annotated with a distance will be assumed to have distance zero—i.e., related under equivalence. For instance, consider the non-relational pairs $(5, 42) : \mathbb{R} \times \mathbb{R}$ and $(1, 42) : \mathbb{R} \times \mathbb{R}$, then

$$((5, 42), (1, 42)) \notin \mathcal{V}[\![\mathbb{R} \times \mathbb{R}]\!]^\rho$$

since $5 \neq 1$. However, non-relational pairs with relational components can be related in a metric relation; this is:

$$((\mathbf{N} 5, 42), (\mathbf{N} 1, 42)) \in \mathcal{V}[\![\text{Rel } 7 \mathbb{R} \times \mathbb{R}]\!]^\rho$$

since $|5 - 1| \leq 7$ and $42 \equiv 42$

With these logical relations, we establish the notion of type soundness via the fundamental lemma of logical relations (i.e., well-typed terms are related to themselves), which also corresponds to the metric preservation theorem [33].

Theorem B.1 (Metric preservation). *Let a well-typed expression $\Psi; \mathcal{C}; \Gamma \vdash e : \tau$ be given. For any ρ for which $\Psi \subseteq \text{dom}(\rho)$ and $\rho \models \mathcal{C}$; suppose γ_1, γ_2 are two substitutions for Γ such that $(\gamma_1, \gamma_2) \in \mathcal{S}[\![\Gamma]\!]^\rho$, then we have $(\gamma_1 \vdash e, \gamma_2 \vdash e) \in \mathcal{E}[\![\tau]\!]_\Gamma^\rho$.*

Proof. By induction on the typing derivations of e . The cases for non-relational terms are standard. A full-blown proof of all relational and non-relational terms can be found in the accompanying material. \square

B.3.1 Sensitivity by Parametricity

In this section, we explore the connection between parametricity and function sensitivity. In a nutshell, we show that by assigning a relational interpretation for λ_{SPAR} types, a proof of function sensitivity can be derived from such interpretation given that λ_{SPAR} is parametric on distances.

The concept of *parametricity* [39] refers to a generic property of programming languages supporting parametric polymorphism. This property captures the intuition that every instance of a polymorphic function should behave the same. Wadler’s key observation is that by interpreting types as relations, instead of sets, one can produce useful theorems about programs directly from their types. For instance, when considering any polymorphic list-transformation function $r : \forall A.[A] \rightarrow [A]$, one can use parametricity to obtain the following (free) theorem:

$$\forall f \, xs. \text{map } f \, (r \, xs) \equiv r \, (\text{map } f \, xs)$$

This theorem tells us insightful information about the way r interacts with its input: it works on the structure of the input list in a way that is independent of the elements of the list. Formally, parametricity states that any closed term e of type τ is related to itself under a relational interpretation of its types, this is:

$$\Psi; \mathcal{C}; \emptyset \vdash e : \tau \Rightarrow (e, e) \in \llbracket \tau \rrbracket \quad (\text{B.1})$$

with $\llbracket \tau \rrbracket \in \tau \times \tau$ denoting the relational interpretation for τ . In the previous section, we defined a set of logical relations providing a metric interpretation to our types. Then, if we define $\llbracket \tau \rrbracket$ as $\mathcal{E}\llbracket \tau \rrbracket_\emptyset^\emptyset$, the parametricity lemma corresponds to the fundamental lemma of logical relations—i.e, metric preservation Theorem B.1—where the substitutions γ_1 and γ_2 are empty; thus trivially related.

With this in mind, we argue that given a parametric closed function f , which for any distance i , it has type $\text{Rel } i \, \sigma_1 \rightarrow \text{Rel } (k * i) \, \sigma_2$, we can prove that f satisfies k -sensitivity via metric preservation. Concretely, when we consider parametric functions such as:

$$\Psi, i; \mathcal{C}; \emptyset \vdash f : \text{Rel } i \, \sigma_1 \rightarrow \text{Rel } (k * i) \, \sigma_2$$

the previous statement describes a sensitivity soundness theorem of the form:

Theorem B.2 (Sensitivity soundness). *Given a function f and type variable i , it holds that*

$$\Psi, i; \mathcal{C}; \emptyset \vdash f : \text{Rel } i \, \sigma_1 \rightarrow \text{Rel } (k * i) \, \sigma_2 \Rightarrow f \text{ is } k\text{-sensitive}$$

Proof. Recall Definition B.1 stating that a function f is k -sensitive if the distance between its outputs is bounded by k times the distance between its inputs—for whatever distance they might have. In terms of our logical relations, k -sensitivity for

closed functions can be expressed as follows:

$$\begin{aligned}
& \forall (\Psi, i), \mathcal{C}, \rho, v_1, v_2. (\Psi, i) \subseteq \text{dom}(\rho) \wedge \rho \models \mathcal{C} \\
& \quad \wedge (v_1, v_2) \in \mathcal{V}[\![\text{Rel } i \sigma_1]\!]^\rho \\
& \quad \Rightarrow (\cdot \vdash f @ v_1, \cdot \vdash f @ v_2) \in \mathcal{E}[\![\text{Rel } (k * i) \sigma_2]\!]^\rho_\emptyset
\end{aligned} \tag{B.2}$$

By parametricity (i.e, metric preservation) over f we know:

$$\begin{aligned}
& \forall (\Psi, i), \mathcal{C}, \rho. (\Psi, i) \subseteq \text{dom}(\rho) \wedge \rho \models \mathcal{C} \\
& \quad \Rightarrow (\cdot \vdash f, \cdot \vdash f) \in \mathcal{E}[\![\text{Rel } i \sigma_1 \rightarrow \text{Rel } (k * i) \sigma_2]\!]^\rho_\emptyset
\end{aligned} \tag{B.3}$$

Now, let's expand the conclusion of this implication:

$$\begin{aligned}
& (\cdot \vdash f, \cdot \vdash f) \in \mathcal{E}[\![\text{Rel } i \sigma_1 \rightarrow \text{Rel } (k * i) \sigma_2]\!]^\rho_\emptyset \\
& \equiv \langle \text{By definition of } \mathcal{E}[\![_\emptyset]^\rho \rangle \\
& \quad \forall f_1, f_2. \cdot \vdash f \Downarrow f_1 \wedge \cdot \vdash f \Downarrow f_2 \\
& \quad \Rightarrow (f_1, f_2) \in \mathcal{V}[\![\text{Rel } i \sigma_1 \rightarrow \text{Rel } (k * i) \sigma_2]\!]^\rho \\
& \equiv \langle \text{By determinism of } (_ \Downarrow _) \text{ with } \cdot \vdash f \Downarrow \langle \lambda x.e \mid \cdot \rangle \rangle \\
& \quad (\langle \lambda x.e \mid \cdot \rangle, \langle \lambda x.e \mid \cdot \rangle) \in \mathcal{V}[\![\text{Rel } i \sigma_1 \rightarrow \text{Rel } (k * i) \sigma_2]\!]^\rho \\
& \equiv \langle \text{By definition of } \mathcal{V}[\![_\emptyset]^\rho \text{ with } \Gamma = \emptyset \rangle \\
& \quad \forall v_1, v_2. (v_1, v_2) \in \mathcal{V}[\![\text{Rel } i \sigma_1]\!]^\rho \\
& \quad \Rightarrow ([x := v_1] \vdash e, [x := v_2] \vdash e) \in \mathcal{E}[\![\text{Rel } (k * i) \sigma_2]\!]^\rho_{x:\text{Rel } i \sigma_1} \\
& \equiv \langle \text{By definition of } \mathcal{E}[\![_{x:\text{Rel } i \sigma_1}]^\rho \text{ and } (_ \Downarrow _) \rangle \\
& \quad \forall v_1, v_2. (v_1, v_2) \in \mathcal{V}[\![\text{Rel } i \sigma_1]\!]^\rho \\
& \quad \Rightarrow (\cdot \vdash \lambda x.e @ v_1, \cdot \vdash \lambda x.e @ v_2) \in \mathcal{E}[\![\text{Rel } (k * i) \sigma_2]\!]^\rho_\emptyset \\
& \equiv \langle \text{Since } \cdot \vdash f \Downarrow \langle \lambda x.e \mid \cdot \rangle \text{ and } \cdot \vdash \lambda x.e \Downarrow \langle \lambda x.e \mid \cdot \rangle \rangle \\
& \quad \forall v_1, v_2. (v_1, v_2) \in \mathcal{V}[\![\text{Rel } i \sigma_1]\!]^\rho \\
& \quad \Rightarrow (\cdot \vdash f @ v_1, \cdot \vdash f @ v_2) \in \mathcal{E}[\![\text{Rel } (k * i) \sigma_2]\!]^\rho_\emptyset
\end{aligned} \tag{B.4}$$

When rewriting (B.3) with (B.4) we obtain:

$$\begin{aligned}
& \forall (\Psi, i), \mathcal{C}, \rho, v_1, v_2. (\Psi, i) \subseteq \text{dom}(\rho) \wedge \rho \models \mathcal{C} \\
& \quad \wedge (v_1, v_2) \in \mathcal{V}[\![\text{Rel } i \sigma_1]\!]^\rho \\
& \quad \Rightarrow (\cdot \vdash f @ v_1, \cdot \vdash f @ v_2) \in \mathcal{E}[\![\text{Rel } (k * i) \sigma_2]\!]^\rho_\emptyset
\end{aligned}$$

Which is no less than the definition (B.2) of a k -sensitive function expressed in terms of logical relations. \square

Our reasoning showcases the usefulness of parametricity as a technique for obtaining insightful theorems about parametric functions. In particular, we have shown that by giving a metric interpretation to λ_{SPAR} types, sensitivity soundness (Theorem B.2) follows directly from the metric preservation Theorem B.1. In conclusion, we have shown that tracking programs' distances and allowing parametricity on distance terms are sufficient conditions to prove sensitivity soundness.

```

data Rel (d :: Nat) a
  -- Numbers
  lit :: Int → Rel d Int
  (:+) :: Rel d1 Int → Rel d2 Int
        → Rel (d1+d2) Int
  -- Pairs
  (:*) :: Rel d1 a → Rel d2 b → Rel (d1+d2) b
  -- Vectors
  Nil :: Rel d (Vec 0 a)
  (:>) :: Rel da a → Rel dv (Vec m a)
        → Rel (da+dv) (Vec (m+1) a)
  -- Sub-typing
  up :: Rel d a → Rel (d+c) a
  -- Sensitivity type synonym
  type Sen (k :: Nat) a b =
    ∀ (d :: Nat) . Rel d a → Rel (k*d) b
  -- Using sensitivity functions
  run :: (Rf a, Rf b) ⇒ Sen k a b → (a → b)

```

Figure B.4: SPAR API

B.4 λ_{SPAR} as a library

In this section, we present SPAR, the software library that realizes the calculus from Section B.2. As captured by our main result in Section B.3.1, the library relies on Haskell’s *parametrically polymorphic functions* to compute sensitivity for user-defined functions. Besides polymorphism, our implementation uses some of the advanced features of Haskell’s type-system to facilitate the usage of the library—we defer explanations about such features to Section B.5.

SPAR is a domain-specific language[21] (DSL) embedded in Haskell, that means that the language constructs are given as a library of ordinary Haskell functions. Not all language constructs need to be part of the SPAR core language. One of the great benefits of an embedded language is the ability to use the host language to create programs. For instance, it is possible to leverage any of Haskell’s high-order functions to compactly describe functions and prove their sensitivity.

The full API of SPAR is presented in Figure B.4. Type **Rel** is indexed by a type-level natural number. Unlike our calculus, SPAR only considers natural numbers as distances. This limitation mainly arises from Haskell’s type-system not being powerful enough to represent real numbers, and their operations, at the type level. Primitives **number** and **(+)** correspond to the relational numbers and addition from our calculus. Similarly, primitive **(*)**, **Nil**, and **(>)** correspond to relational pairs, and relational vectors, respectively. As we mentioned in Section B.2, the construction

of pairs (\vdash) and vectors $(\vdash>)$ work similarly when tracking distances at the type-level, i.e., $d_a + d_b$ and $d_a + d_v$, respectively. Primitive `up` encodes the sub-typing rule from our calculus. The type synonym `Sen` directly refers to functions from `a` to `b` with proven sensitivity `k`, where the proof comes from the explicit use of parametric polymorphism. Finally, function `run` takes a function with proven sensitivity `k` and obtains the underlying function so that it can be executed by the host language—we defer the explanation about how to use it and its constraints until the next section. In what follows, we will see some examples showing how SPAR can support reasoning about the sensitivity of complex higher-order recursive functions. Extending SPAR with primitives such as `laplace` and `exponential` mechanism would allow us to capture canonical differentially-private algorithms (e.g., histograms and cumulative sensitive functions); we explore how SPAR could be used under this setting.

B.4.1 Vectors

We start with an example where developers can write functions, and by doing so, *discover* and *provide a proof* about their sensitivity. Obtaining proof is merely to convince the type-checker. We focus on analyzing the sensitivity of the well-known `map` function:

```
map :: (a → b) → [a] → [b]
```

which takes a function from `a` to `b` (`a → b`), a list of elements of type `a` (`[a]`) and applies the function to each element to obtain a list of elements of type `b` (`[b]`). What is the sensitivity of the `map` function? To answer that question, we proceed to implement `map` using SPAR. We assume that the argument of the `map` function is of sensitivity `k`, that is,

```
smap :: Sen k a b → ...
```

(The `...` denotes some irrelevant code to the point being made.) Since we need to reason about how the output changes with respect to the input, we make `smap` to take and return relational values.

```
smap :: Sen k a b → Rel d (Vec m a) → ...
```

What should it be the distance of the relational output? At this point, we can simply write the function and put a type variable and wait for the type-system to complain.

```
smap :: Sen k a b → Rel d (Vec m a) → Rel x (Vec m b)
smap f Nil = Nil
smap f (x:>xs) = f x:>smap f xs
```

The type-system indeed complains and says

```
Could not deduce: ((k * da) + dv) ~ x arising
from a use of ‘:>’
from the context: (Vec m a ~ Vec (m1 + 1) a1,
                  (da + dv) ~ x)
...
```

Observe that the type-system is helping us to figure out the shape of x , i.e., it should be, at least, $k \cdot da + dv$ where we know that $da + dv$ is equal (unifies) to x . With such information, we can propose that x increases its value to $k \cdot da + k \cdot dv$, which is equivalent to $k \cdot (da + dv)$ or simply $k \cdot d$.

```
smap :: Sen k a b → Rel d (Vec m a)
      → Rel (k*d) (Vec m b)
```

With this type-signature `smap` type-checks! and we can proceed to use the `Sen` type-synonym to indicate that `smap f` has sensitivity k given that f has sensitivity k :

```
smap :: Sen k a b → Sen k (Vec m a) (Vec m b)
```

As shown by this example, SPAR leverages Haskell's type-system to provide some restricted support for sensitivity inference. To the best of our knowledge, sensitivity inference has not been addressed in previous work, e.g., [33, 22, 29].

Similarly, we can show that summing elements of a vector or appending two vectors are operations with sensitivity 1:

```
ssum :: Sen 1 (Vec m Int) Int
ssum Nil      = lit 0
ssum (x:>xs) = x.+ssum xs
```

```
sappend :: Sen 1 (Vec m Int, Vec n Int) (Vec (m+n) Int)
sappend (Nil :*:ys) = up ys
sappend ((x:>xs):*:ys) = x:>sappend (xs:*:ys)
```

Observe that `sappend` utilizes the primitive `up` when pattern-matching on the pair `Nil:*:ys`. If the primitive `up` were to be removed, the type-system would complain:

```
* Could not deduce: d2 ~ d
from the context:
  ((Vec m Int, Vec n Int) ~ (a, b),
   (d1 + d2) ~ d)
...
```

From the error message, the type-system is hinting to us that `d2` should somehow unify with `d`. To achieve that, we can use the primitive `up` to transform a relational value with distance `d2` into one with distance `d2+d1`, which is equal to `d`. As before, the type-system aids programmers in proving sensitivity.

SPAR can be used to prove the sensitivity of many list-related functions:

```
-- Left-fold
sfoldl :: Sen 1 (a, b) b → Sen 1 (b, Vec m a) b
-- Right-fold
sfoldr :: Sen 1 (a, b) b → Sen 1 (b, Vec m a) b
-- Concatenation
```

```

sCurry
  :: Sen k (a, b) c → Rel d1 a → Rel d2 b
  → Rel (k*(d1+d2)) c
sUncurry
  :: (∀ d1 d2 . Rel d1 a → Rel d2 b
  → Rel (k*(d1+d2)) c)
  → Sen k (a, b) c

```

Figure B.5: API for currying

```

sconcat :: Sen 1 (Vec m (Vec n Int)) (Vec (m*n) Int)
-- Zipping / unzipping
szip     :: Sen 1 (Vec n a, Vec n b) (Vec n (a, b))
sunzip   :: Sen 1 (Vec n (a, b)) (Vec n a, Vec n b)
szipWith :: Sen k (a, b) c
          → Sen k (Vec n a, Vec n b) (Vec n c)

```

From the type-signature of `sfoldl`, for instance, we can assert that left-folding of a vector with a function of sensitivity 1 has also sensitivity 1.

B.4.2 Currying

The attentive reader might have noticed that `sappend`, and most of the functions above are given in an *uncurried* form, i.e., they take all of their arguments in an n -tuple. For instance, function `sappend` takes the two vectors to be concatenated in a pair:

```

sappend :: Rel d (Vec m Int, Vec n Int)
        → Rel d (Vec (m+n) Int)

```

However, programmers can feel more comfortable writing curried versions of functions:

```

sappend' :: Rel d1 (Vec m Int) → Rel d2 (Vec n Int)
          → Rel (d1+d2) (Vec (m+n) Int)
sappend' Nil ys = up ys
sappend' (x:>xs) ys = x:>sappend' xs ys

```

While `sappend'` type-checks, to use the `Sen` type-synonym, we need to uncurry it first.

Figure B.5 shows the type signatures of functions `sCurry` and `sUncurry`, which are *derived operations* in our library, i.e., they are simply implemented using `SPAR` API and standard Haskell's primitives. With function `sUncurry`, we can simply uncurry `sappend'` for then using the type-synonym `Sen`:

```

sappend'' :: Sen 1 (Vec m Int, Vec n Int)
              (Vec (m+n) Int)
sappend'' = sUncurry sappend'

```

B.4.3 Sorting

To conclude this section, we show how to implement insertion sort and obtain a proof about its sensitivity. However, doing that requires considering the conditional swap, or *cswp*, operation introduced by Fuzz: “... it takes in a pair, and outputs the same pair, swapped if necessary so that the first component is no larger than the second one.” [33]. This primitive outsources from the program the ability to compare pairs’ elements.

Primitive *cswp* has the following type:

```

cswp :: Sen 1 (Int, Int) (Int, Int)

```

Similar to Fuzz, our calculus is not powerful enough to encode *cswp* and we need to consider it as an add-on primitive to our API. The implementation of *cswp* requires comparing elements of type *Rel d Int* to determine when flipping the pair. With *cswp* in place, we can implement insert sort.

```

16 insert :: Rel da Int → Rel dv (Vec n Int)
17         → Rel (da+dv) (Vec (n+1) Int)
18 insert x Nil      = x:>Nil
19 insert x (y:>ys) =
20   case cswp (x:*y) of
21     a1:*a2 → a1:>insert a2 ys
22 sort :: Sen 1 (Vec n Int) (Vec n Int)
23 sort Nil      = Nil
24 sort (x:>xs) = insert x (sort xs)

```

Lines 16–21 implements function *insert*, which adds an element to a sorted list. Lines 20 shows that *cswp* takes the pair *x:*y*, and the execution continues with the ordered pair *a1:*a2*. With function *insert* in place, we prove that insert sort has sensitivity 1 as expected—see line 22.

B.4.4 Beyond sensitivity

Even though SPAR was envisioned as an *intermediate step* towards creating a system for differentially-private algorithms; in this section, we explore how a few extensions will allow SPAR to be integrated into Haskell-based DP frameworks.

Finite sets are useful collection types that can be added to SPAR’ API together with canonical primitive operations (listed below). As presented in Fuzz, the distance between sets is determined by the Hamming metric.

```

bag :: Set a → Rel n (Set a)

```

```

union, intersect, difference :: Ord a ⇒ Rel n (Set a, Set a) → Rel n (Set a)
size :: Rel n (Set a) → Rel n Int
splitBag :: (a → Bool) → Rel n (Set a)
           → Rel n (Set a, Set a)

```

Now, if there were a function implementing the Laplace mechanism where the aggregation to be computed is as a SPAR k -sensitive function, and the dataset is a relational set $\text{Rel } n \text{ (Set db)}$; then, the noise to be added could be sampled from a Laplace distribution with mean 0 and scale $k * n / \epsilon$; thus satisfying ϵ -DP. Such a function would be typed as:

```

laplace :: ∀ k n db . ε → Sen k (Set db) Int
         → Rel n (Set db) → IO Double

```

With these extensions, we would be able to create canonical differentially-private algorithms such as $m * \epsilon$ -DP histograms (see below), where m is the number of partitions or buckets.

```

dp_hist :: ∀ n . [Bucket] → ε
         → Rel n (Set Int) → IO [(Bucket, Double)]
dp_hist [] _eps _set = return []
dp_hist (c : cs) eps set =
  case splitBag@n@Int (λz → c ≥ z) set of
    cBucket*:rest →
      do countC ← laplace@1 eps size cBucket
         countRest ← dp_hist cs eps rest
         return ((c, countC) : countRest)

```

Even though there are some technical challenges in implementing the hypothesized mechanism (outside this work's scope), this example illustrates how SPAR's proof of sensitivity and distance tracking could be utilized for deploying differentially-private user-defined algorithms.

B.5 Implementation

In this section, we describe some of our design decisions and insights gained while realizing our calculus in Haskell. We expect that these insights could help in implementing SPAR in other programming languages.

B.5.1 Equality for type-level natural numbers

To compute distances, SPAR *relies heavily on the abilities of the type-system to manipulate, either concretely or symbolically, type-level natural number expressions and decide their equality*. We utilize the Glasgow Haskell Compiler (GHC) and some plugins extensions for injecting new axioms into GHC's type equality relation while not breaking type safety.

To illustrate this point, we consider the propositional equality $a \sim b$ which asserts that type a is the same as type b . This equality constraint has the *only* constructor $\text{Ref1} :: a \sim a$. When the occurrences of $\text{Ref1} :: a \sim b$ type-check, the compiler has done its job unifying a with b .

```
isEqualInt :: Int ~: Int
isEqualInt = Ref1
```

However, when handling type-level natural numbers, we would like types $1+n$ and $n+1$ to match—as we do in math. For instance, if we write the following code, it will be rejected by GHC’s type-system.

```
-- It does not type-check!
isEqualPlus :: (1+n) ~: (n+1)
isEqualPlus = Ref1
```

We use the GHC plugin `TypeLits.Normalize` to extend the compiler with more equality of type-level natural number expressions. This extension works by normalizing type-level natural numbers, variables, and arithmetic expressions (e.g., $+$, $-$, and $*$) to (sort of) sum-of-products representation, and then performing syntactic equality.¹ Once this extension is activated, the following types are considered equal.

```
-- These examples type-check with TypeLits.Normalize
isEqualPlus :: (1+n) ~: (n+1)
isEqualPlus = Ref1
isEqualMult :: (n*k) ~: (k*n)
isEqualMult = Ref1
```

While an improvement, for some of our examples presented in Section B.4, GHC needs to understand better the interplay between the associativity of type-level operators like $+$ and unification. For instance, we would like GHC to assert that $(n1+(n2+n3)) \sim (a+b)$ under the hypothesis $(n1+n2) \sim (a+c)$ —refer as H1—and $(c+n3) \sim b$ —refer as H2. The proof that such unification exists is as follows:

```
-- by associativity of +
(n1+(n2+n3)) ~: (n1+n2)+n3
-- by hypothesis H1
(n1+(n2+n3)) ~: (a+c)+n3
-- by associativity of +
(a+c)+n3 ~: a+(c+n3)
-- by hypothesis H2
a+(c+n3) ~: a+b
```

We want the compiler to do the proof for us. To achieve that, we utilize the Thoralf plugin [31] to translate the unification constraints to queries to an external SMT

¹<https://hackage.haskell.org/package/ghc-typelits-natnormalise-0.7.6/docs/GHC-TypeLits-Normalise.html>

```

-- Introduction of relational terms
data Rel d a where
  -- Numbers
  LInt :: Int → Rel d Int
  -- Pairs with evidence
  Pair :: d ~ (da+db) ⇒ Rel da a → Rel db b
        → Rel d (a, b)
  ... -- other cases
-- Elimination of relational terms
pattern d1*:d2 = Pair d1 d2

```

Figure B.6: Deep embedded primitives

solver. In that manner, it becomes possible for GHC to prove the equality we described above.

```

-- It type-checks with the Thoralf plugin
isEqual :: (n1+n2) :~: (a+c)      -- H1
        → (c+n3) :~: b          -- H2
        → (n1+(n2+n3)) :~: (a+b) -- thesis
isEqual Refl Refl = Refl

```

The use of SMT solvers is justified by its accessibility in Haskell, and the simplicity of SPAR’s generated constraints. However, we remark that SMT solvers are not *fundamental* to our approach, but a constraint solving procedure is needed instead since, like other systems for differential privacy, SPAR is quantitative in nature. In other words, one could write a simple automated solving procedure removing the dependency on solvers. Lastly, if SPAR were implemented in a language with a more expressive type-system, e.g., like dependent types [17, 30] or liquid types [35, 38], we would expect minor (if any) dependencies on external add-ons components to the compiler.

B.5.2 SPAR as an embedded DSL

SPAR is a *deep* eDSL when it comes to implementing introduction and elimination constructs from our calculus, but a *shallow* eDSL when it comes to operations among relational values [1]. The deeply embedded part of our DSL does not perform any actual computation; instead, they result in a data structure representing the relational values being constructed. Figure B.6 shows the implementation of `Rel d a` for the introduction of relational numbers and pairs. What appears on the left-hand side of \Rightarrow are *type constraints*. They can be seen as static demands for the types involved on the right-hand side of it. Constructor `LInt` is parametric on the distance `d`. Constructor `Pair` introduces the same distance constraint as our calculus in Section B.2 (i.e., $d_a + d_b \sim d$). When it comes to elimination rules, we utilize

```

(.,+) :: Rel d1 Int → Rel d2 Int → Rel (d1+d2) Int
(LInt n).+(LInt m) = LInt (n+m)

```

Figure B.7: Shallow embedded operations

```

run :: (Rf a, Rf b) ⇒ Sen k a b → a → b
run f = fromDist . f . toDist
class Rf a where
  toDist :: a → Rel 0 a
  fromDist :: Rel 0 a → a
instance Rf Int where
  ...
instance (Rf a, Rf b) ⇒ Rf (a, b) where
  ...
instance (Rf a) ⇒ Rf (Vec l a) where
  ...

```

Figure B.8: Executing functions with proven sensitivity

pattern-synonyms [32] which allows us to expose specific constructors and hide others. For instance, the pattern-synonym `(:*)` enables to construct and deconstruct (i.e., pattern-matching) over pairs—thus enabling to conveniently write programs like $\lambda(d1:*.d2) \rightarrow d1.+d1.+d2$. Importantly, we do not enable pattern-matching on relational numbers. If we did, programmers could write functions that break SPAR’s guarantees:

```

isTen :: Rel d Int → Rel 0 Int
isTen (Lit 10) = Lit 1
isTen _       = Lit 0

```

According to the type signature, this function returns a constant, i.e., a value at distance 0. However, the function definition can return outputs at a distance 1.

Figure B.7 gives an example of a relational operation written in a shallow embedded manner, where the semantics of `+` is given based on the host language semantics of `+`. This design choice avoids SPAR from introducing pattern-matching on relational operations (like `+`) and just focusing on supporting pattern-matching on basic constructors (like `pairs`)—recall Figure B.6.

B.5.3 Executing functions

Figure B.8 shows the implementation of `run`—symbol `.` denotes function composition. To implement it, it is necessary to take a value of type `a` in the host semantics

and *reify* it into SPAR—see function `toDist`—, apply the function, and *reflect* the result back into the host language—see function `fromDist`.

The type class constraint `Rf a` limits the types which can be mapped into SPAR. Since the sensitivity for $f :: \text{Sen } k \ a \ b$ has been proven to be k , functions `toDist` and `fromDist` simply apply f with relational values at distance `0`. The type classes instances declare that integers (`Int`), pairs `((a,b))`, and vectors (`Vec n a`) can be reified to and reflected from SPAR. To give a concrete example, we utilize function `smap` from Section B.4 to run a function which duplicates the content of a vector.

```
duplicate :: Vec m Int → Vec m Int
duplicate = run (smap f)
  where f :: Sen 2 Int Int
        f x = x.+x
>run duplicate (VCons 1 (VCons 2 (VCons 3 VNil)))
(VCons 2 (VCons 4 (VCons 6 VNil)))
```

Constructor `VNil` is used to produce empty vectors, i.e., those of length `0`; `VCons` is a function that inserts an element of type `a` in a vector of length `n`, yielding a new vector of length `n+1`.

Developers should be careful using function `run`. This function *forgets* about distances among relational values, and as such, it should never be used inside functions computing on relational values. In other words, `run` is safe to be used only as a *top-level primitive*—and there are standard ways to enforce that, e.g., by using Safe Haskell [36].

B.6 Discussion

Non-termination Our use of logical relations in Section B.3 states that if two computations over metrically related inputs *do both terminate*, then their outputs are metrically related. Since our calculus has no fixed-point primitive of the kind **fix** of type $\tau \rightarrow \tau$; our formal guarantees align with our formalization. Nevertheless, adding such primitive demands the use of the well-known mechanism of step-indexed logical relations [4], which makes possible to prove the fundamental theorem of logical relations even in the presence of fixpoints. We leave as future work on how to extend our formalization to address abnormal termination—a good starting point is to consider the mitigations techniques proposed in Fuzz [24].

Branching on relational values As shown in Section B.5.2, enabling branching on relational terms is problematic—a well-known limitation shared in many related work (e.g., [22, 29, 2]). We foresee a possible manner to overcome this limitation by adopting the program continuity verification framework by Chaudhuri et al. [12]. This framework characterizes how a small perturbation to the input variables of a given branch condition can cause the control to flow to exercise different branches, which could lead to a syntactically divergent behavior. Conceptually, this approach is based on a set of syntax-directed proof rules collecting constraints which

are then off-loaded to an SMT-solver. In this light, since SPAR is syntax-directed (module primitive up) and also uses an SMT-solver in the background, we expect to adopt such results in our setting. To prove k -Lipschitz continuity (equivalent to k -sensitivity), however, Chaudhuri et al.’s work requires to collect constraints that assert that each branch is k -Lipschitz continuous with respect to the inputs and outputs variables. One of the challenges we foreseen is to elegantly capture, at the type-level, such constraints when using a case-statement and still being able to provide SPAR as a library.

Distance using real numbers Our implementation of λ_{SPAR} only considers distances as natural numbers. This design decision is based on the limitations of Haskell’s type system. Dependently-typed languages—like Coq, Idris, and Agda—can easily support (axiomatic² or constructive [23, 13]) encodings of real numbers at the type-level in a natural way. To illustrate this point, we reformulate the type-signature of `lit` from Figure B.4 in a type-dependent fashion:

$$\text{lit} : \forall \{A : \text{Set}\} \{d : \mathbb{R}\} \text{Rel } d \ A$$

where \mathbb{R} is the type for representing real numbers and d is a term of that type. Different from SPAR, the type `Rel` is indexed by a term-level real number. There is a series of work on formulating parametricity with dependent types [8, 9, 10]. In this light, we expect the our soundness results also hold in such a setting—an interesting direction for future work.

B.7 Related work

Sensitivity by Linear types Several works have studied techniques to reason about program sensitivity by typing, most of which in the context of differential privacy. An early approach is the work by Reed and Pierce [33]. They designed an indexed linear type system for differential privacy where types explicitly track sensitivities thanks to types of the form $!_r A \multimap B$. In their work, this type can only be assigned to terms representing functions from A to B which have sensitivity less than r . Functions of these forms could be turned into differentially private programs by adding noise carefully calibrated to r . The type system by Reed and Pierce [33] was implemented in the language Fuzz which was also extended with a timed runtime to avoid side channels with respect to the differential privacy guarantee [24]. Automated type inference for this type system was studied by D’Antoni *et al.* [14], and its semantics foundation was studied by Azevedo de Amorim *et al.* [16]. Fuzz was further extended in several directions: Eigner and Maffei [20] extended Fuzz to reason about distributed data and differentially private security protocols. Gaboardi *et al.* [22] extended Fuzz’s type checker by means of a simple form of dependent types. Winograd-Cort *et al.* [43] extended Fuzz type checker and runtime system to an adaptive framework. Zhang *et al.* [46] extended the ideas of Fuzz to a three-level logic for reasoning about sensitivity for primitives that are not captured in Fuzz.

²Like in <https://coq.inria.fr/library/Coq.Reals.Raxioms.html>

Azevedo de Amorim *et al.* [15] add to Fuzz more general rules for reasoning about the sensitivity of programs returning probability distributions.

Different from Fuzz’s line of work, our approach does not require the use of linear types.

Other type-based approaches Near *et al.* [29] designed the language Duet to support other notion of differential privacy. The Duet approach is based on the design of a two-layers language. The underlying layer is similar to Fuzz, and the other layer is a linear type system without annotations for sensitivities. This second layer does not impose constraints on the distance of elements, and it can hence support approximate differential privacy and other relaxation of differential privacy. However, this approach limits the support that Duet can provide for higher order functions. Toro *et al.* [37] further extended this approach by combining linear types with contextual effects. The resulting system supports different notions of differential privacy and higher-order functions. However, the type system now has to track both linear and contextual effect information. Concurrently to SPAR’s development, Abuah *et al.* [2] designed a type system (named SOLO) useful for reasoning about sensitivity without requiring linear types—a goal that aligns with ours. Their type system is also embedded in Haskell and leverages polymorphism for some specific parts of the implementation. While SOLO’s and SPAR’s approaches are comparable, they differ on the following key aspects. Firstly, `textscSolo`’s formalism and soundness proofs are based on a monomorphic calculus, even though parametric polymorphism is highlighted as essential to support higher-order functions. In contrast, λ_{SPAR} is polymorphic and is used to prove that sensitivity soundness can be deduced from parametricity. Secondly, SOLO attaches sensitivities to base types while SPAR attaches distances. By tracking distances instead of sensitivities, SPAR returns the notion of sensitivity to functions (as formally defined) instead of values. This differentiating factor allows SPAR’s users to reason about the sensitivity of their functions, and the distance of their values in a—arguably—more intuitive and fundamental way. Lastly, Thanks to the use of parametricity for characterizing sensitivity, SPAR’s users can type and implement recursive higher-order programs; more importantly, all the formal guarantees (i.e., sensitivity-soundness) hold for these user-defined functions. On the other hand, SOLO adds recursive functions as primitives since the structure of sensitive recursive types (such as lists) is not accessible to the programmers.

Relational type-systems Several works have also explored how to reason about sensitivity using relational type systems. These approaches are quite different from Fuzz’s and the one presented in this work. This line of work was pioneered by Barthe *et al.* [7] and Zhang and Kifer [44] and further extended afterward, e.g. [6, 42, 41]. Relational type systems are not readily available and require specialized implementations. Moreover, they are not easy to use, also for specialists.

Program analysis Other approaches to reason about program sensitivity were based on program analysis. To reason about the continuity of programs, Chaudhuri *et al.* [12] designed a program analysis tracking the usage of variables and giving

an upper bound on the program’s sensitivity. Johnson *et al.* [25] proposed a static analysis to track sensitivities of queries in a SQL-like system. Specifically, their approach combines abstract interpretation ideas with the quantitative tracking of sensitivities for basic operations like Count and Sum. Abua *et al.* [3] designed a dynamic sensitivity analysis which tracks sensitivity and metric information at the values level. This dynamic analysis is used to guarantee differential privacy in an adaptive setting, similar to the one explored in Adaptive Fuzz [43].

Parametricity Studies of parametricity and its variants abound in the literature. It all started with the seminal paper by Reynolds [34], where the polymorphic semantics of System F’s types is captured in a suitable model. Wadler then popularized this result as a tool to deduce theorems for polymorphic function [39]. Our main result to prove the sensitivity of functions can be seen as a theorem arising for parametrically polymorphic functions on distances. Other security conditions, like non-interference, have also been proven using parametricity [11, 5]. To be best of our knowledge, we are the first ones to show that sensitivity proofs can be obtained *for free* by parametricity. There exists a series of work on obtaining parametricity results for dependent-typed languages [8, 9, 10]—which constitutes interesting results when realizing λ_{SPAR} in languages like Agda, Coq, or Idris.

B.8 Conclusions

We have presented λ_{SPAR} , a sound calculus that uses parametricity to prove the sensitivity of functions by type-checking. The calculus is simple, and that is its beauty. We also showed how to implement the calculus as the library SPAR for the programming language Haskell—where the total library consists of 360 lines of code. We expect that SPAR serves as a basis for providing a light-weight verification tool to certify, for instance, the sensitivity of arbitrary functions in the exponential mechanism [19] for differential privacy: if the score function has type `Sen k (Vec n a, Range) Int`, then we can be sure that its sensitivity is `k` regardless of who wrote that function.

Appendix

B.A SPAR's complete syntax

$$i \in \text{dvar} \quad j \in \text{lvar} \quad x \in \text{evar} \quad r \in \mathbb{R}_{\geq 0} \quad n \in \mathbb{R}$$

$$\begin{aligned} \tau \in \text{type} & ::= \sigma \mid \tau \times \tau \mid \vec{\tau}_l \mid \tau \rightarrow \tau \mid \text{Rel } d \sigma \\ \sigma \in \text{type} & ::= \mathbb{R} \mid \sigma \times \sigma \mid \vec{\sigma}_l \\ d \in \text{dist} & ::= i \mid r \mid d + d \mid d * d \\ l \in \text{length} & ::= j \mid 0 \mid l + 1 \end{aligned}$$

$$\begin{aligned} e \in \text{expr} & ::= x \mid n \mid e + e \mid (e, e) \mid \mathbf{fst} \, e \mid \mathbf{snd} \, e \\ & \quad \mid \lambda x. e \mid e @ e \mid [] \mid e :: e \\ & \quad \mid \mathbf{case} \, e \mathbf{of} \{ [] . e \} \{ (x :: xs) . e \} \\ & \quad \mid \mathbf{N} \, e \mid e + e \mid (e, e) \\ & \quad \mid \mathbf{let} \, (x, y) = e \mathbf{in} \, e \mid [] \mid e :: e \\ & \quad \mid \mathbf{case} \, e \mathbf{of} \{ [] . e \} \{ (x :: xs) . e \} \end{aligned}$$

$$\begin{aligned} v \in \text{val} & ::= n \mid (v, v) \mid \langle \lambda x. e \mid \gamma \rangle \mid [] \mid v :: v \\ & \quad \mid \mathbf{N} \, v \mid (v, v) \mid [] \mid v :: v \end{aligned}$$

$$\begin{aligned} \Gamma \in \text{tenv} & ::= \emptyset \mid \Gamma, x : \tau \\ \Psi \in \text{denv} & ::= \emptyset \mid (\Psi_d, \Psi_l) \\ \Psi_d \in \text{denv} & ::= \emptyset \mid \Psi_d, i \\ \Psi_l \in \text{lenv} & ::= \emptyset \mid \Psi_l, j \\ \mathcal{C} \in \text{cenv} & ::= \emptyset \mid (\mathcal{C}_d, \mathcal{C}_l) \\ \mathcal{C}_d \in \text{cdenv} & ::= \emptyset \mid d = d \mid \mathcal{C}_d \wedge \mathcal{C}_d \\ \mathcal{C}_l \in \text{clenv} & ::= \emptyset \mid l = l \mid \mathcal{C}_l \wedge \mathcal{C}_l \end{aligned}$$

B.B Typing system

$$\begin{array}{c}
\text{D.VAR} \\
\frac{i \in \Psi.d}{\Psi \vdash i} \\
\\
\text{D.REAL} \\
\frac{r \in \mathbb{R}_{\geq 0}}{\Psi \vdash r} \\
\\
\text{D.ADD } \dot{\star} \text{ D.PROD} \\
\frac{\Psi \vdash d_1 \quad \Psi \vdash d_2 \quad \star \in \{+, *\}}{\Psi \vdash d_1 \star d_2}
\end{array}$$

(a) Typing rules for distance expressions

$$\begin{array}{c}
\text{L.VAR} \\
\frac{j \in \Psi.l}{\Psi \vdash j} \\
\\
\text{L.ZERO} \\
\frac{}{\Psi \vdash 0} \\
\\
\text{L.SUCC} \\
\frac{\Psi \vdash l}{\Psi \vdash l + 1}
\end{array}$$

(b) Typing rules for length expressions

$$\begin{array}{c}
\text{ST.REFL} \\
\frac{}{\Psi; \mathcal{C} \models \tau \sqsubseteq \tau} \\
\\
\text{ST.TRANS} \\
\frac{\Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_2 \quad \Psi; \mathcal{C} \models \tau_2 \sqsubseteq \tau_3}{\Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_3} \\
\\
\text{ST.}\times \\
\frac{\Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_3 \quad \Psi; \mathcal{C} \models \tau_2 \sqsubseteq \tau_4}{\Psi; \mathcal{C} \models \tau_1 \times \tau_2 \sqsubseteq \tau_3 \times \tau_4} \\
\\
\text{ST.VEC} \\
\frac{\Psi; \mathcal{C} \models l = l \quad \Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_2}{\Psi; \mathcal{C} \models \vec{\tau}_{1l} \sqsubseteq \vec{\tau}_{2l}} \\
\\
\text{ST.}\rightarrow \\
\frac{\Psi; \mathcal{C} \models \tau_3 \sqsubseteq \tau_1 \quad \Psi; \mathcal{C} \models \tau_2 \sqsubseteq \tau_4}{\Psi; \mathcal{C} \models \tau_1 \rightarrow \tau_2 \sqsubseteq \tau_3 \rightarrow \tau_4} \\
\\
\text{ST.REL} \\
\frac{\Psi; \mathcal{C} \models d_1 \dot{\leq} d_2}{\Psi; \mathcal{C} \models \text{Rel } d_1 \sigma \sqsubseteq \text{Rel } d_2 \sigma}
\end{array}$$

(c) Sub-typing rules

Figure B.9: Typing rules for several components

Where determining if d_1 is less or equal than d_2 given the environment Ψ and constraints \mathcal{C} is defined as:

$$\Psi; \mathcal{C} \models d_1 \dot{\leq} d_2 \iff \forall \rho. \Psi \subseteq \text{dom}(\rho) \wedge \rho \models \mathcal{C} \Rightarrow \llbracket d_1 \rrbracket_\rho \leq \llbracket d_2 \rrbracket_\rho$$

$$\begin{array}{c}
\text{SAT.EMPTY} \\
\frac{}{\rho \models \emptyset} \\
\\
\text{SAT.EQ} \\
\frac{\llbracket a \rrbracket_\rho \equiv \llbracket b \rrbracket_\rho}{\rho \models a = b} \\
\\
\text{SAT.CONJ} \\
\frac{\rho \models \mathcal{C}_1 \quad \rho \models \mathcal{C}_2}{\rho \models \mathcal{C}_1 \wedge \mathcal{C}_2}
\end{array}$$

Figure B.10: Satisfiability relation

$$\begin{array}{c}
\text{T.VAR} \quad \frac{x : \tau \in \Gamma}{\Psi; \mathcal{C}; \Gamma \vdash x : \tau} \quad \text{T.NUM} \quad \frac{}{\Psi; \mathcal{C}; \Gamma \vdash n : \mathbb{R}} \quad \text{T.ADD} \quad \frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \mathbb{R} \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \mathbb{R}}{\Psi; \mathcal{C}; \Gamma \vdash e_1 + e_2 : \mathbb{R}} \\
\\
\text{T.PAIR} \quad \frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \tau_1 \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \tau_2}{\Psi; \mathcal{C}; \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \quad \text{T.FST} \text{ } \dot{\neq} \text{ T.SND} \quad \frac{\Psi; \mathcal{C}; \Gamma \vdash e : \tau_1 \times \tau_2}{\Psi; \mathcal{C}; \Gamma \vdash \mathbf{fst} \, e : \tau_1} \\
\quad \quad \quad \Psi; \mathcal{C}; \Gamma \vdash \mathbf{snd} \, e : \tau_2 \\
\\
\text{T.LAM} \quad \frac{\Psi; \mathcal{C}; \Gamma, x : \tau_1 \vdash e : \tau_2}{\Psi; \mathcal{C}; \Gamma \vdash \lambda x. e : \tau_1 \rightarrow \tau_2} \quad \text{T.APP} \quad \frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \tau_1}{\Psi; \mathcal{C}; \Gamma \vdash e_1 @ e_2 : \tau_2} \\
\\
\text{T.NIL} \quad \frac{j \notin \Psi.l}{\Psi, j; \mathcal{C} \wedge j = 0; \Gamma \vdash [] : \vec{\tau}_j} \quad \text{T.CONS} \quad \frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \tau \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \vec{\tau}_l \quad j \notin \Psi.l}{\Psi, j; \mathcal{C} \wedge j = (l+1); \Gamma \vdash e_1 :: e_2 : \vec{\tau}_j} \\
\\
\text{T.CASE} \quad \frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \vec{\tau}_l \quad \Psi; \mathcal{C} \wedge l = 0; \Gamma \vdash e_2 : \tau' \quad \Psi, j; \mathcal{C} \wedge l = (j+1); \Gamma, x : \tau, xs : \vec{\tau}_j \vdash e_3 : \tau' \quad j \notin \Psi.l}{\Psi; \mathcal{C}; \Gamma \vdash \mathbf{case} \, e_1 \, \mathbf{of} \, \{[] . e_2\} \{(x :: xs) . e_3\} : \tau'}
\end{array}$$

Figure B.11: Typing rules for non-relational terms

Figure B.11 presents the typing rules for non-relational terms. The typing rules for these cases are just like those of the simply typed lambda calculus. As expected, the rules do not interact with the environment of distance variables Ψ_d nor the set of distance constraints \mathcal{C}_d .

$$\begin{array}{c}
\text{T.NUM}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e : \mathbb{R} \quad \text{vars}(d) \subseteq \Psi.d}{\Psi; \mathcal{C}; \Gamma \vdash \mathbf{N} e : \text{Rel } d \mathbb{R}}
\\[10pt]
\text{T.ADD}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \mathbb{R} \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \mathbb{R} \quad i \notin \Psi.d}{\Psi, i; \mathcal{C} \wedge i = (d_1 + d_2); \Gamma \vdash e_1 + e_2 : \text{Rel } i \mathbb{R}}
\\[10pt]
\text{T.PAIR}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \sigma_1 \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \sigma_2 \quad i \notin \Psi.d}{\Psi, i; \mathcal{C} \wedge i = (d_1 + d_2); \Gamma \vdash (e_1, e_2) : \text{Rel } i (\sigma_1 \times \sigma_2)}
\\[10pt]
\text{T.LET}_R \\
\frac{\begin{array}{c} \Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d (\sigma_1 \times \sigma_2) \\ \Psi, i_1, i_2; \mathcal{C} \wedge d = (i_1 + i_2); \Gamma, x : \text{Rel } i_1 \sigma_1, y : \text{Rel } i_2 \sigma_2 \vdash e_2 : \tau \\ i_1, i_2 \notin \Psi.d \end{array}}{\Psi; \mathcal{C}; \Gamma \vdash \mathbf{let} (x, y) = e_1 \mathbf{in} e_2 : \tau}
\\[10pt]
\text{T.NIL}_R \\
\frac{j \notin \Psi.l \quad \text{vars}(d) \subseteq \Psi.d}{\Psi, j; \mathcal{C} \wedge j = 0; \Gamma \vdash [] : \text{Rel } d \vec{\sigma}_j}
\\[10pt]
\text{T.CONSR}_R \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \sigma \quad \Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \vec{\sigma}_l \quad i \notin \Psi.d \quad j \notin \Psi.l}{\Psi, i, j; \mathcal{C} \wedge j = (l + 1) \wedge i = (d_1 + d_2); \Gamma \vdash e_1 :: e_2 : \text{Rel } i \vec{\sigma}_j}
\\[10pt]
\text{T.CASE}_R \\
\frac{\begin{array}{c} \Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d \vec{\sigma}_l \quad \Psi; \mathcal{C} \wedge l = 0; \Gamma \vdash e_2 : \tau \\ \Psi, i_1, i_2, j; \mathcal{C} \wedge l = (j + 1) \wedge d = (i_1 + i_2); \Gamma, x : \text{Rel } i_1 \sigma, xs : \text{Rel } i_2 \vec{\sigma}_j \vdash e_3 : \tau \\ i_1, i_2 \notin \Psi.d \quad j \notin \Psi.l \end{array}}{\Psi; \mathcal{C}; \Gamma \vdash \mathbf{case} e_1 \mathbf{of} \{ [] . e_2 \} \{ (x :: xs) . e_3 \} : \tau}
\\[10pt]
\text{T.}\sqsubseteq \\
\frac{\Psi; \mathcal{C}; \Gamma \vdash e : \tau_1 \quad \Psi; \mathcal{C} \models \tau_1 \sqsubseteq \tau_2}{\Psi; \mathcal{C}; \Gamma \vdash e : \tau_2}
\end{array}$$

Figure B.12: Typing rules for relational terms

B.C Semantics

B.C.1 Operational

$\frac{\text{E.VAR}}{\gamma \vdash x \Downarrow \gamma(x)}$	$\frac{\text{E.NUM}}{\gamma \vdash n \Downarrow n}$	$\frac{\text{E.ADD} \quad \gamma \vdash e_1 \Downarrow v_1 \quad \gamma \vdash e_2 \Downarrow v_2}{\gamma \vdash e_1 + e_2 \Downarrow v_1 + v_2}$
$\frac{\text{E.APP} \quad \gamma \vdash e_1 \Downarrow \langle \lambda x. e_3 \mid \gamma' \rangle \quad \gamma \vdash e_2 \Downarrow v_2 \quad \gamma'[x := v_2] \vdash e_3 \Downarrow v_3}{\gamma \vdash e_1 @ e_2 \Downarrow v_3}$		
$\frac{\text{E.LAM}}{\gamma \vdash \lambda x. e \Downarrow \langle \lambda x. e \mid \gamma \rangle}$	$\frac{\text{E.PAIR} \ \& \ \text{E.PAIR}_R \quad \gamma \vdash e_1 \Downarrow v_1 \quad \gamma \vdash e_2 \Downarrow v_2}{\gamma \vdash (e_1, e_2) \Downarrow (v_1, v_2) \quad \gamma \vdash (e_1, e_2) \Downarrow (v_2, v_2)}$	$\frac{\text{E.FST} \ \& \ \text{E.SND}}{\gamma \vdash e \Downarrow (v_1, v_2) \quad \gamma \vdash \mathbf{fst} \ e \Downarrow v_1 \quad \gamma \vdash \mathbf{snd} \ e \Downarrow v_2}$
$\frac{\text{E.LET}_R \quad \gamma \vdash e_1 \Downarrow (v_1, v_2) \quad \gamma[x := v_1][y := v_2] \vdash e_2 \Downarrow v}{\gamma \vdash \mathbf{let} \ (x, y) = e_1 \ \mathbf{in} \ e_2 \Downarrow v}$		$\frac{\text{E.NIL} \ \& \ \text{E.NIL}_R}{\gamma \vdash [] \Downarrow [] \quad \gamma \vdash [] \Downarrow []}$
$\frac{\text{E.CONS} \ \& \ \text{E.CONS}_R \quad \gamma \vdash e_1 \Downarrow v_1 \quad \gamma \vdash e_2 \Downarrow v_2}{\gamma \vdash e_1 :: e_2 \Downarrow v_1 :: v_2 \quad \gamma \vdash e_1 :: e_2 \Downarrow v_1 :: v_2}$	$\frac{\text{E.CASE-NIL} \quad \gamma \vdash e_1 \Downarrow [] \quad \gamma \vdash e_2 \Downarrow v_2}{\gamma \vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[] . e_2\} \{(x :: xs). e_3\} \Downarrow v_2}$	
$\frac{\text{E.CASE-CONS} \quad \gamma \vdash e_1 \Downarrow v_1 :: vs_1 \quad \gamma[x := v_1][xs := vs_1] \vdash e_3 \Downarrow v_3}{\gamma \vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[] . e_2\} \{(x :: xs). e_3\} \Downarrow v_3}$		
$\frac{\text{E.CASE-NIL}_R \quad \gamma \vdash e_1 \Downarrow [] \quad \gamma \vdash e_2 \Downarrow v_2}{\gamma \vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[] . e_2\} \{(x :: xs). e_3\} \Downarrow v_2}$		
$\frac{\text{E.CASE-CONS}_R \quad \gamma \vdash e_1 \Downarrow v_1 :: vs_1 \quad \gamma[x := v_1][xs := vs_1] \vdash e_3 \Downarrow v_3}{\gamma \vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[] . e_2\} \{(x :: xs). e_3\} \Downarrow v_3}$		$\frac{\text{E.NUM}_R}{\gamma \vdash e \Downarrow v \quad \gamma \vdash \mathbf{N} \ e \Downarrow \mathbf{N} \ v}$
$\frac{\text{E.ADD}_R \quad \gamma \vdash e_1 \Downarrow \mathbf{N} \ v_1 \quad \gamma \vdash e_2 \Downarrow \mathbf{N} \ v_2}{\gamma \vdash e_1 + e_2 \Downarrow \mathbf{N} \ (v_1 + v_2)}$		

Figure B.13: Evaluation rules

Figure B.13 shows the big step operational semantics. When evaluated, relational terms just reduce as if they were non-relational ones. The only non-standard, but not surprising, evaluation rule is the one for destructing relational pairs (**let** $(x, y) = e_1$ **in** e_2), where the substitution gets extended with the values resulting from evaluating the pair.

B.C.2 Distance and length interpretation

Given a general assignment $\rho \in (\text{dvar} \rightarrow \mathbb{R}_{\geq 0}, \text{lvar} \rightarrow \mathbb{N})$ we provide an interpretation for *dist* and *length* expressions (Figure B.14). Accessing ρ 's components is denoted as $\rho.d$ and $\rho.l$.

$$\begin{array}{ll}
 \llbracket _ \rrbracket_{\rho.d} \in \text{dist} \rightarrow (\text{dvar} \rightarrow \mathbb{R}_{\geq 0}) \rightarrow \mathbb{R}_{\geq 0} & \llbracket _ \rrbracket_{\rho.l} \in \text{length} \rightarrow (\text{lvar} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \\
 \llbracket i \rrbracket_{\rho.d} = \rho.d(i) & \llbracket j \rrbracket_{\rho.l} = \rho.l(j) \\
 \llbracket r \rrbracket_{\rho.d} = r & \llbracket 0 \rrbracket_{\rho.l} = 0 \\
 \llbracket d_1 \star d_2 \rrbracket_{\rho.d} = \llbracket d_1 \rrbracket_{\rho.d} \star \llbracket d_2 \rrbracket_{\rho.d} \quad \star \in \{+, *\} & \llbracket l + 1 \rrbracket_{\rho.l} = \llbracket l \rrbracket_{\rho.l} + 1
 \end{array}$$

(a) dist as non-negative real values

(b) length as natural numbers

Figure B.14: Distance and length interpretation

We use the general function $\llbracket _ \rrbracket_{\rho}$ to refer to both interpretations at once. We'll use $\llbracket _ \rrbracket_{\rho.d}$ and $\llbracket _ \rrbracket_{\rho.l}$ only when it is necessary to disambiguate between the two.

B.D Logical relations

Our presentation here differs slightly from the definitions given on the paper; in particular, the relational interpretation of functions $(\mathcal{V}[\tau_1 \times \tau_2]^\rho)$ and open terms $(\mathcal{V}[\tau_1 \times \tau_2]^\rho)$ lacks quantification on typing context Γ , and substitutions γ_1 and γ_2 . Nonetheless, these instantiations are trivial in the proof and do not affect their outcome.

B.D.1 Definitions

--Metric relations

$$\begin{aligned}
\mathcal{D}[\mathbb{R}]_d^\rho &= \{(\mathbf{N} \ r_1, \mathbf{N} \ r_2) \mid |r_1 - r_2| \leq \llbracket d \rrbracket_{\rho, \mathbf{d}}\} \\
\mathcal{D}[\sigma_1 \times \sigma_2]_d^\rho &= \{((v_{11}, v_{21}), (v_{12}, v_{22})) \mid \exists d_1, d_2. \rho, \mathbf{d} \models d = d_1 + d_2 \\
&\quad \wedge (v_{11}, v_{12}) \in \mathcal{D}[\sigma_1]_{d_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\sigma_2]_{d_2}^\rho\} \\
\mathcal{D}[\vec{\sigma}_l]_d^\rho &= \text{case } \llbracket l \rrbracket_{\rho, 1} \text{ of} \\
&\quad 0 \longrightarrow \{([\], [\])\} \\
&\quad l' \longrightarrow \{(v_{11} :: v_{21}, v_{12} :: v_{22}) \mid \exists d_1, d_2. \rho, \mathbf{d} \models d = d_1 + d_2 \\
&\quad \wedge (v_{11}, v_{12}) \in \mathcal{D}[\sigma]_{d_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\vec{\sigma}_{(l'-1)}]_{d_2}^\rho\}
\end{aligned}$$

--Relational interpretation of values

$$\begin{aligned}
\mathcal{V}[\mathbb{R}]^\rho &= \{(v_1, v_2) \mid v_1 \equiv v_2\} \\
\mathcal{V}[\tau_1 \times \tau_2]^\rho &= \{((v_{11}, v_{21}), (v_{12}, v_{22})) \mid (v_{11}, v_{12}) \in \mathcal{V}[\tau_1]^\rho \\
&\quad \wedge (v_{21}, v_{22}) \in \mathcal{V}[\tau_2]^\rho\} \\
\mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho &= \{(\langle \lambda x. e_1 \mid \gamma_1 \rangle, \langle \lambda x. e_2 \mid \gamma_2 \rangle) \mid \forall v_1, v_2. (v_1, v_2) \in \mathcal{V}[\tau_1]^\rho \\
&\quad \Rightarrow (\gamma_1 \vdash \lambda x. e_1 @ v_1, \gamma_2 \vdash \lambda x. e_2 @ v_2) \in \mathcal{E}[\tau_2]_\Gamma^\rho\} \\
\mathcal{V}[\vec{\tau}_l]^\rho &= \text{case } \llbracket l \rrbracket_{\rho, 1} \text{ of} \\
&\quad 0 \longrightarrow \{([\], [\])\} \\
&\quad l' \longrightarrow \{(v_{11} :: v_{21}, v_{12} :: v_{22}) \mid (v_{11}, v_{12}) \in \mathcal{V}[\tau]^\rho \\
&\quad \wedge (v_{21}, v_{22}) \in \mathcal{V}[\vec{\tau}_{(l'-1)}]^\rho\} \\
\mathcal{V}[\text{Rel } d \ \sigma]^\rho &= \mathcal{D}[\sigma]_d^\rho
\end{aligned}$$

--Relational interpretation of terms

$$\begin{aligned}
\mathcal{E}[\tau]_\Gamma^\rho &= \{(\gamma_1 \vdash e_1, \gamma_2 \vdash e_2) \mid \forall v_1, v_2. \gamma_1 \vdash e_1 \Downarrow v_1 \wedge \gamma_2 \vdash e_2 \Downarrow v_2 \\
&\quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\tau]^\rho\}
\end{aligned}$$

--Relational interpretation of contexts

$$\begin{aligned}
\mathcal{S}[\Gamma]^\rho &= \{(\gamma_1, \gamma_2) \mid \text{dom}(\gamma_1) \equiv \text{dom}(\gamma_2) \equiv \text{dom}(\Gamma) \\
&\quad \wedge \forall (x : \tau). x \in \text{dom}(\Gamma) \Rightarrow (\gamma_1(x), \gamma_2(x)) \in \mathcal{V}[\tau]^\rho\}
\end{aligned}$$

B.E Proof of metric preservation and accompanying lemmas

Lemma B.3 (Exchange).

$$\forall \Psi, \mathcal{C}, G, G', \tau_1, \tau_2, \tau, x_1, x_2, e. \Gamma = G, x_1 : \tau_1, x_2 : \tau_2, G' \quad (\text{H1})$$

$$\wedge \Delta = G, x_2 : \tau_2, x_1 : \tau_1, G' \quad (\text{H2})$$

$$\wedge \Psi; \mathcal{C}; \Gamma \vdash e : \tau \quad (\text{H3})$$

$$\Rightarrow \Psi; \mathcal{C}; \Delta \vdash e : \tau \quad (\text{C})$$

Proof. By induction on the typing derivations of e

- **Case [T.VAR]:** $\Psi; \mathcal{C}; \Gamma \vdash x : \tau$

Observe that Γ and Δ associate the same variables with the same types, in other words $\text{dom}(\Gamma) \equiv \text{dom}(\Delta)$ and $\forall x \in \text{dom}(\Gamma). \Gamma(x) \equiv \Delta(x)$, then it must be the case that

$$x : \tau \in \Gamma \Rightarrow x : \tau \in \Delta \quad (\text{F1})$$

By (H3) and [T.VAR] we know that $x : \tau \in \Gamma$ (F2), then by (F2) and (F1) we get $x : \tau \in \Delta$ (F3). Finally, by [T.VAR] and (F3) we obtain $\Psi; \mathcal{C}; \Delta \vdash x : \tau$

- **Case [T.LAM]:** $\Psi; \mathcal{C}; \Gamma \vdash \lambda x. e' : (\tau'_1 \rightarrow \tau'_2)$

Consider the IH in $\Psi; \mathcal{C}; \Gamma, x : \tau'_1 \vdash e' : \tau'_2$:

$$\forall G^*, G'^*, \tau_1^*, \tau_2^*, x_1^*, x_2^*. \Gamma, x : \tau'_1 = G^*, x_1^* : \tau_1^*, x_2^* : \tau_2^*, G'^* \quad (\text{P1})$$

$$\wedge \Delta, x : \tau'_1 = G^*, x_2^* : \tau_2^*, x_1^* : \tau_1^*, G'^* \quad (\text{P2})$$

$$\Rightarrow \Psi; \mathcal{C}; \Delta, x : \tau'_1 \vdash e' : \tau'_2 \quad (\text{IH}')$$

Let $G^* = G, G'^* = G', x : \tau'_1, \tau_1^* = \tau_1, \tau_2^* = \tau_2, x_1^* = x_1$, and $x_2^* = x_2$; this way (P1) and (P2) are proven by (H1) and (H2), respectively. Having proven both premises we can conclude that:

$$\Psi; \mathcal{C}; \Delta, x : \tau'_1 \vdash e' : \tau'_2 \quad (\text{IH}')$$

Then, by (IH') and [T.LAM] we obtain $\Psi; \mathcal{C}; \Delta \vdash \lambda x. e' : (\tau'_1 \rightarrow \tau'_2)$

The rest of the cases are trivial. □

Lemma B.4 (Weakening).

$$\forall \Psi, \mathcal{C}, \tau, \tau', e, y. \Psi; \mathcal{C}; \Gamma \vdash e : \tau \quad (\text{H1})$$

$$\wedge y \notin \Gamma \quad (\text{H2})$$

$$\Rightarrow \Psi; \mathcal{C}; \Gamma, y : \tau' \vdash e : \tau \quad (\text{C})$$

Proof. By induction on the typing derivations of e

- **Case [T.VAR]:** $\Psi; \mathcal{C}; \Gamma \vdash x : \tau$

Consider the context $\Delta = \Gamma, y : \tau'$ such that $\Gamma \subseteq \Delta$. By (H1) and [T.VAR] we know that $x : \tau \in \Gamma$, then, by definition of $(_ \in _)$ and $(_ \subseteq _)$ it must be the case that $x : \tau \in \Delta$ (F1).

With (F1) and [T.VAR] we obtain $\Psi; \mathcal{C}; \Delta \vdash x : \tau$ which is equivalent to $\Psi; \mathcal{C}; \Gamma, y : \tau' \vdash x : \tau$ by definition of Δ .

- **Case [T.LAM]:** $\Psi; \mathcal{C}; \Gamma \vdash \lambda x. e' : (\tau_1 \rightarrow \tau_2)$

By inductive hypothesis in $\Psi; \mathcal{C}; \Gamma, x : \tau_1 \vdash e' : \tau_2$ we get:

$$\Psi; \mathcal{C}; \Gamma, x : \tau_1, y : \tau' \vdash e' : \tau_2 \quad (\text{IH}')$$

By (IH') and exchange (Lemma B.3) we have that:

$$\Psi; \mathcal{C}; \Gamma, y : \tau', x : \tau_1 \vdash e' : \tau_2$$

then, by [T.VAR] we obtain:

$$\Psi; \mathcal{C}; \Gamma, y : \tau' \vdash \lambda x. e' : (\tau_1 \rightarrow \tau_2)$$

The rest of the cases are trivial. □

Lemma B.5. *Relation $\mathcal{D}[\![_]\!]_d^\rho$ is preserved under interpretation of distance terms. Concretely, if $(v_1, v_2) \in \mathcal{D}[\![\sigma]\!]_d^\rho$ then $(v_1, v_2) \in \mathcal{D}[\![\sigma]\!]_{[d]_{\rho, d}}^{(\emptyset, \rho, l)}$*

Proof. By structural induction on σ

- **Case $\sigma = \mathbb{R}$**

$$\text{Goal: } \forall v_1, v_2, \rho, d. (v_1, v_2) \in \mathcal{D}[\![\mathbb{R}]\!]_d^\rho \Rightarrow (v_1, v_2) \in \mathcal{D}[\![\mathbb{R}]\!]_{[d]_{\rho, d}}^{(\emptyset, \rho, l)}$$

$$\begin{aligned} & (v_1, v_2) \in \mathcal{D}[\![\mathbb{R}]\!]_d^\rho \\ \equiv & \langle \text{By cases in } v_1, v_2 \rangle \\ & (\mathbb{N} n_1, \mathbb{N} n_2) \in \mathcal{D}[\![\mathbb{R}]\!]_d^\rho \\ \equiv & \langle \text{By def of } \mathcal{D}[\![_]\!]_d^\rho \text{ at } \mathbb{R} \rangle \\ & |n_1 - n_2| \leq [d]_{\rho, d} \end{aligned} \quad (\text{H})$$

Let $[d]_{\rho, d} = r$ (F1) then (H) can be rewritten as $|n_1 - n_2| \leq r$ (H')

Now, lets consider our objective

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{D}[\![\mathbb{R}]\!]_{\rho, d}^{(\emptyset, \rho, 1)} \\
& \equiv \langle \text{By (F1)} \rangle \\
& (v_1, v_2) \in \mathcal{D}[\![\mathbb{R}]\!]_r^{(\emptyset, \rho, 1)} \\
& \equiv \langle \text{By cases in } v_1, v_2 \rangle \\
& (\mathbf{N} \, n_1, \mathbf{N} \, n_2) \in \mathcal{D}[\![\mathbb{R}]\!]_r^{(\emptyset, \rho, 1)} \\
& \equiv \langle \text{By def of } \mathcal{D}[\![_d]^\rho \text{ at } \mathbb{R} \rangle \\
& |n_1 - n_2| \leq \llbracket r \rrbracket_\emptyset \\
& \equiv \langle \text{By def of } \llbracket r \rrbracket_{\rho, d} \rangle \\
& |n_1 - n_2| \leq r
\end{aligned}$$

Which is proven by (H')

- **Case** $\sigma = \sigma_1 \times \sigma_2$

$$\text{Goal: } \forall v_1, v_2, \rho, d. (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^\rho \Rightarrow (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_{\rho, d}^{(\emptyset, \rho, 1)}$$

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^\rho \\
& \equiv \langle \text{By cases in } v_1, v_2 \rangle \\
& ((v_{11}, v_{21}), (v_{12}, v_{22})) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^\rho \\
& \equiv \langle \text{By def of } \mathcal{D}[\![_d]^\rho \text{ at } (- \times -) \rangle \\
& \exists d_1, d_2. \rho, d \models d = d_1 + d_2 \tag{H1} \\
& \wedge (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d_1}^\rho \tag{H2} \\
& \wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d_2}^\rho \tag{H3}
\end{aligned}$$

Consider the IHs in σ_1 and σ_2

$$\forall v'_{11}, v'_{12}, \rho_1, d'_1. (v'_{11}, v'_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d'_1}^{\rho_1} \Rightarrow (v'_{11}, v'_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{\rho_1, d}^{(\emptyset, \rho_1, 1)} \tag{IH1}$$

$$\forall v'_{21}, v'_{22}, \rho_2, d'_2. (v'_{21}, v'_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d'_2}^{\rho_2} \Rightarrow (v'_{21}, v'_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{\rho_2, d}^{(\emptyset, \rho_2, 1)} \tag{IH2}$$

Let $v'_{11} = v_{11}, v'_{12} = v_{12}, \rho_1 = \rho$, and $d'_1 = d_1$; then by (H2) and (IH1) we get:

$$(v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{\rho, d_1}^{(\emptyset, \rho, 1)} \tag{IH1'}$$

Similarly, let $v'_{21} = v_{21}, v'_{22} = v_{22}, \rho_2 = \rho$, and $d'_2 = d_2$; then, by (H3) and (IH2) we get:

$$(v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{\rho, d_2}^{(\emptyset, \rho, 1)} \tag{IH2'}$$

Take $\llbracket d_1 \rrbracket_{\rho, d} = r_1$ (F1) and $\llbracket d_2 \rrbracket_{\rho, d} = r_2$ (F2), then (IH1') and (IH2') can be rewritten as:

$$(v_{11}, v_{12}) \in \mathcal{D}[\llbracket \sigma_1 \rrbracket_{r_1}^{(\emptyset, \rho, 1)}] \wedge (v_{21}, v_{22}) \in \mathcal{D}[\llbracket \sigma_2 \rrbracket_{r_2}^{(\emptyset, \rho, 1)}] \quad (\text{IH}')$$

Moreover, by (H1) we know that $\rho, d \models d = d_1 + d_2$, then by definition of $(_ \models _)$ it must be the case that $\llbracket d \rrbracket_{\rho, d} \equiv \llbracket d_1 \rrbracket_{\rho, d} + \llbracket d_2 \rrbracket_{\rho, d}$, which, by (F1) and (F2), is equivalent to $\llbracket d \rrbracket_{\rho, d} \equiv r_1 + r_2$ (F3).

Recall our objective

$$\begin{aligned} & (v_1, v_2) \in \mathcal{D}[\llbracket \sigma_1 \times \sigma_2 \rrbracket_{\llbracket d \rrbracket_{\rho, d}}^{(\emptyset, \rho, 1)}] \\ & \equiv \langle \text{By cases in } v_1, v_2 \text{ together with (F3)} \rangle \\ & \quad ((v_{11}, v_{21}), (v_{12}, v_{22})) \in \mathcal{D}[\llbracket \sigma_1 \times \sigma_2 \rrbracket_{(r_1 + r_2)}^{(\emptyset, \rho, 1)}] \\ & \equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_d^\rho \text{ at } (_ \times _) \rangle \\ & \quad \exists d_1^*, d_2^*. \emptyset \models r_1 + r_2 = d_1^* + d_2^* \quad (\text{G1}) \\ & \quad \wedge (v_{11}, v_{12}) \in \mathcal{D}[\llbracket \sigma_1 \rrbracket_{d_1^*}^{(\emptyset, \rho, 1)}] \quad (\text{G2}) \\ & \quad \wedge (v_{21}, v_{22}) \in \mathcal{D}[\llbracket \sigma_2 \rrbracket_{d_2^*}^{(\emptyset, \rho, 1)}] \quad (\text{G3}) \end{aligned}$$

Take $d_1^* = r_1$ and $d_2^* = r_2$, then (G1) is trivially proven by definition of $(_ \models _)$. Lastly, (G2) and (G3) are directly proven by (IH').

□

Lemma B.6. *If $i \notin \mathbf{FV}(\sigma)$ and $i \notin \text{vars}(d)$ then $(v_1, v_2) \in \mathcal{D}[\llbracket \sigma \rrbracket_d^\rho] \equiv (v_1, v_2) \in \mathcal{D}[\llbracket \sigma \rrbracket_d^{\rho \setminus i}$. In other words:*

$$\forall \sigma, v_1, v_2, \rho, i, d. i \notin \mathbf{FV}(\sigma) \quad (\text{H1})$$

$$\wedge i \notin \text{vars}(d) \quad (\text{H2})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{D}[\llbracket \sigma \rrbracket_d^\rho] \equiv (v_1, v_2) \in \mathcal{D}[\llbracket \sigma \rrbracket_d^{\rho \setminus i}] \quad (\text{C})$$

Proof. By structural induction on σ .

- **Case $\sigma = \mathbb{R}$**

Let's analyze the left-hand side of the equivalence:

$$\begin{aligned} & (v_1, v_2) \in \mathcal{D}[\llbracket \mathbb{R} \rrbracket_d^\rho] \\ & \equiv \langle \text{By cases in } v_1, v_2 \rangle \\ & \quad (\mathbf{N} \ n_1, \mathbf{N} \ n_2) \in \mathcal{D}[\llbracket \mathbb{R} \rrbracket_d^\rho] \\ & \equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_d^\rho \text{ at } \mathbb{R}] \rangle \\ & \quad |n_1 - n_2| \leq \llbracket d \rrbracket_{\rho, d} \end{aligned}$$

By (H2) we know that $i \notin \text{vars}(d)$, then $\llbracket d \rrbracket_{\rho, d} \equiv \llbracket d \rrbracket_{\rho, d \setminus i}$ (F1) by definition of $\llbracket _ \rrbracket_{\rho, d}$. Then

$$\begin{aligned}
& |n_1 - n_2| \leq \llbracket d \rrbracket_{\rho, d} \\
& \equiv \langle \text{By (F1)} \rangle \\
& |n_1 - n_2| \leq \llbracket d \rrbracket_{\rho, d \setminus i} \\
& \equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_d^\rho \text{ at } \mathbb{R} \text{ and def of } \rho] \rangle \\
& (\mathbf{N} \, n_1, \mathbf{N} \, n_2) \in \mathcal{D}[\llbracket \mathbb{R} \rrbracket_d^{\rho \setminus i} \\
& \equiv \langle \text{By cases in } v_1, v_2 \text{ we got } v_1 = \mathbf{N} \, n_1, v_2 = \mathbf{N} \, n_2 \rangle \\
& (v_1, v_2) \in \mathcal{D}[\llbracket \mathbb{R} \rrbracket_d^{\rho \setminus i}
\end{aligned}$$

• **Case** $\sigma = \sigma_1 \times \sigma_2$

Our goal is

$$\forall v_1, v_2, \rho, i, d. i \notin \mathbf{FV}(\sigma_1 \times \sigma_2) \quad (\text{H1})$$

$$\wedge i \notin \text{vars}(d) \quad (\text{H2})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{D}[\llbracket \sigma_1 \times \sigma_2 \rrbracket_d^\rho \equiv (v_1, v_2) \in \mathcal{D}[\llbracket \sigma_1 \times \sigma_2 \rrbracket_d^{\rho \setminus i} \quad (\text{C})$$

And the IH in σ_1 and σ_2 are of the form:

$$\forall v'_{11}, v'_{12}, \rho_1, i_1, d'_1. i_1 \notin \mathbf{FV}(\sigma_1) \quad (\text{P1.1})$$

$$\wedge i_1 \notin \text{vars}(d'_1) \quad (\text{P1.2})$$

$$\Rightarrow (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma_1 \rrbracket_{d'_1}^{\rho_1} \equiv (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma_1 \rrbracket_{d'_1}^{\rho_1 \setminus i_1} \quad (\text{IH1})$$

$$\forall v'_{21}, v'_{22}, \rho_2, i_2, d'_2. i_2 \notin \mathbf{FV}(\sigma_2) \quad (\text{P2.1})$$

$$\wedge i_2 \notin \text{vars}(d'_2) \quad (\text{P2.2})$$

$$\Rightarrow (v'_{21}, v'_{22}) \in \mathcal{D}[\llbracket \sigma_2 \rrbracket_{d'_2}^{\rho_2} \equiv (v'_{21}, v'_{22}) \in \mathcal{D}[\llbracket \sigma_2 \rrbracket_{d'_2}^{\rho_2 \setminus i_2} \quad (\text{IH2})$$

Now, to prove (C) we can decompose it in both directions of the equivalence as follows:

$$\forall v_1, v_2, \rho, i, d. i \notin \mathbf{FV}(\sigma_1 \times \sigma_2) \quad (\text{H1.1})$$

$$\wedge i \notin \text{vars}(d) \quad (\text{H1.2})$$

$$\wedge (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^\rho \quad (\text{H1.3})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^{\rho \setminus i} \quad (\text{C1})$$

$$\forall v_1, v_2, \rho, i, d. i \notin \mathbf{FV}(\sigma_1 \times \sigma_2) \quad (\text{H2.1})$$

$$\wedge i \notin \text{vars}(d) \quad (\text{H2.2})$$

$$\wedge (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^{\rho \setminus i} \quad (\text{H2.3})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^\rho \quad (\text{C2})$$

Let's start by proving (C1). Expanding (H1.3) we have:

$$\begin{aligned} & (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^\rho \\ \equiv & \langle \text{By cases in } v_1, v_2 \rangle \\ & ((v_{11}, v_{21}), (v_{12}, v_{22})) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^\rho \\ \equiv & \langle \text{By def of } \mathcal{D}[\![_]\!]_d^\rho \text{ at } (_ \times _) \rangle \\ & \exists d_1, d_2. \rho.d \models d = d_1 + d_2 \quad (\text{H1.3.1}) \\ & \wedge (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d_1}^\rho \quad (\text{H1.3.2}) \\ & \wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d_2}^\rho \quad (\text{H1.3.3}) \end{aligned}$$

Recall the IH in σ_1 (IH1) and σ_2 (IH2). Let $v'_{11} = v_{11}$, $v'_{12} = v_{12}$, $v'_{21} = v_{21}$, $v'_{22} = v_{22}$, $\rho_1 = \rho_2 = \rho$, $i_1 = i_2 = i$, $d'_1 = d_1$, and $d'_2 = d_2$.

By (H1.1) we know that $i \notin \mathbf{FV}(\sigma_1 \times \sigma_2)$, then, it must be the case that $i \notin \mathbf{FV}(\sigma_1)$ and $i \notin \mathbf{FV}(\sigma_2)$, proving (P1.1) and (P2.1), respectively. Moreover, since $i \notin \text{vars}(d)$ (H1.2) and $\rho.d \models d = d_1 + d_2$ (H1.3.1) it must follow that $i \notin \text{vars}(d_1)$ and $i \notin \text{vars}(d_2)$ ¹, proving (P1.2) and (P2.2). Lastly, since $\rho.d \models d = d_1 + d_2$ and $i \notin \text{vars}(d) \cup \text{vars}(d_1) \cup \text{vars}(d_2)$, then $\rho.d \setminus i \models d = d_1 + d_2$ (F1) by definition of $(_ \models _)$.

Having fulfilled the preconditions (P1.1-2) and (P2.1-2) we conclude by (IH1) and (IH2) that:

$$(v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d_1}^\rho \equiv (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d_1}^{\rho \setminus i} \quad (\text{IH'1})$$

$$\wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d_2}^\rho \equiv (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d_2}^{\rho \setminus i} \quad (\text{IH'2})$$

¹To be precise, LR's must be parametric on Ψ and the existentials on relational pairs (and vectors) should be of the form $\exists d_1, d_2. \rho.d \models d = d_1 + d_2 \wedge \text{vars}(d_1) \cup \text{vars}(d_2) \subseteq \Psi.d$. Then the lemma should refer to variable freshness in terms of $i \notin \Psi.d$ instead of $i \notin \mathbf{FV}(\sigma)$. Adapting these changes in the definition over complicates the notation hence it is omitted.

We have

$$\begin{aligned}
& \exists d_1, d_2. \rho.d \models d = d_1 + d_2 \wedge (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d_2}^\rho \\
& \equiv \langle \text{By (F1), (IH'1), and (IH'2)} \rangle \\
& \exists d_1, d_2. \rho.d \setminus i \models d = d_1 + d_2 \wedge (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d_1}^{\rho \setminus i} \wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d_2}^{\rho \setminus i} \\
& \equiv \langle \text{By def of } \mathcal{D}[\![_]\!]_d^\rho \text{ at } (_ \times _) \rangle \\
& ((v_{11}, v_{21}), (v_{12}, v_{22})) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^{\rho \setminus i} \\
& \equiv \langle \text{By cases in } v_1, v_2 \text{ we got } v_1 = (v_{11}, v_{21}), v_2 = (v_{12}, v_{22}) \rangle \\
& (v_1, v_2) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_d^{\rho \setminus i}
\end{aligned}$$

Which is no less than our goal (C1). The proof of (C2) follows the same reasoning as that of (C1).

□

Lemma B.7. *If $i \notin \mathbf{FV}(\tau)$ then $(v_1, v_2) \in \mathcal{V}[\![\tau]\!]^\rho \equiv (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^{\rho \setminus i}$. In other words:*

$$\forall \tau, v_1, v_2, \rho, i. i \notin \mathbf{FV}(\tau) \tag{H}$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^\rho \equiv (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^{\rho \setminus i} \tag{C}$$

Proof. By structural induction on τ .

- **Case** $\tau = \mathbb{R}$

Let's analyze the left-hand side of the equivalence:

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\![\mathbb{R}]\!]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[\![_]\!]^\rho \text{ at } \mathbb{R} \rangle \\
& v_1 \equiv v_2 \tag{LH}
\end{aligned}$$

Similarly, we can inspect the right-hand side of the equivalence

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho \setminus i} \\
& \equiv \langle \text{By def of } \mathcal{V}[\![_]\!]^{\rho \setminus i} \text{ at } \mathbb{R} \rangle \\
& v_1 \equiv v_2 \tag{RH}
\end{aligned}$$

Then it is clear that $(LH) \equiv (RH)$

• **Case** $\tau = \tau_1 \times \tau_2$

Our goal is

$$\forall v_1, v_2, \rho, i. i \notin \mathbf{FV}(\tau_1 \times \tau_2) \quad (\text{H})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^\rho \equiv (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^{\rho \setminus i} \quad (\text{C})$$

And the IH in τ_1 and τ_2 are of the form:

$$\forall v_{11}, v_{12}, \rho_1, i_1. i_1 \notin \mathbf{FV}(\tau_1) \quad (\text{P1})$$

$$\Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\tau_1]\!]^{\rho_1} \equiv (v_{11}, v_{12}) \in \mathcal{V}[\![\tau_1]\!]^{\rho_1 \setminus i_1} \quad (\text{IH1})$$

$$\forall v_{21}, v_{22}, \rho_2, i_2. i_2 \notin \mathbf{FV}(\tau_2) \quad (\text{P2})$$

$$\Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\tau_2]\!]^{\rho_2} \equiv (v_{21}, v_{22}) \in \mathcal{V}[\![\tau_2]\!]^{\rho_2 \setminus i_2} \quad (\text{IH2})$$

Now, to prove (C) we can decompose it in both directions of the equivalence as follows:

$$\forall v_1, v_2, \rho, i. i \notin \mathbf{FV}(\tau_1 \times \tau_2) \quad (\text{H1.1})$$

$$\wedge (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^\rho \quad (\text{H1.2})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^{\rho \setminus i} \quad (\text{C1})$$

$$\forall v_1, v_2, \rho, i. i \notin \mathbf{FV}(\tau_1 \times \tau_2) \quad (\text{H2.1})$$

$$\wedge (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^{\rho \setminus i} \quad (\text{H2.2})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^\rho \quad (\text{C2})$$

Let's start by proving (C1). Expanding (H1.2) we have:

$$\begin{aligned} & (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^\rho \\ & \equiv \langle \text{By cases in } v_1, v_2 \rangle \\ & \quad ((v'_{11}, v'_{21}), (v'_{12}, v'_{22})) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^\rho \\ & \equiv \langle \text{By def of } \mathcal{V}[\![_\cdot]^\rho \text{ at } (_ \times _) \rangle \\ & \quad (v'_{11}, v'_{12}) \in \mathcal{V}[\![\tau_1]\!]^\rho \wedge (v'_{21}, v'_{22}) \in \mathcal{V}[\![\tau_2]\!]^\rho \end{aligned} \quad (\text{H'1.2})$$

Recall the IH in τ_1 (IH1) and τ_2 (IH2). Let $v_{11} = v'_{11}$, $v_{12} = v'_{12}$, $v_{21} = v'_{21}$, $v_{22} = v'_{22}$, $\rho_1 = \rho_2 = \rho$, and $i_1 = i_2 = i$.

By (H1.1) we know $i \notin \mathbf{FV}(\tau_1 \times \tau_2)$, then it must be the case that $i \notin \mathbf{FV}(\tau_1)$ (proving (P1)), and $i \notin \mathbf{FV}(\tau_2)$ (proving (P2)). Having fulfilled the preconditions (P1) and (P2) we can conclude from (IH1) and (IH2) that:

$$(v'_{11}, v'_{12}) \in \mathcal{V}[\tau_1]^\rho \equiv (v'_{11}, v'_{12}) \in \mathcal{V}[\tau_1]^{\rho \setminus i} \quad (\text{IH1}')$$

$$\wedge (v'_{21}, v'_{22}) \in \mathcal{V}[\tau_2]^\rho \equiv (v'_{21}, v'_{22}) \in \mathcal{V}[\tau_2]^{\rho \setminus i} \quad (\text{IH2}')$$

In (H'1.2) we have

$$\begin{aligned} & (v'_{11}, v'_{12}) \in \mathcal{V}[\tau_1]^\rho \wedge (v'_{21}, v'_{22}) \in \mathcal{V}[\tau_2]^\rho \\ & \equiv \langle \text{By (IH'1) and (IH'2)} \rangle \\ & (v'_{11}, v'_{12}) \in \mathcal{V}[\tau_1]^{\rho \setminus i} \wedge (v'_{21}, v'_{22}) \in \mathcal{V}[\tau_2]^{\rho \setminus i} \\ & \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (_ \times _) \rangle \\ & ((v'_{11}, v'_{21}), (v'_{12}, v'_{22})) \in \mathcal{V}[\tau_1 \times \tau_2]^{\rho \setminus i} \\ & \equiv \langle \text{By cases in } v_1, v_2 \text{ we got } v_1 = (v'_{11}, v'_{21}), v_2 = (v'_{12}, v'_{22}) \rangle \\ & (v_1, v_2) \in \mathcal{V}[\tau_1 \times \tau_2]^{\rho \setminus i} \end{aligned}$$

Which is no less than our goal (C1). The proof of (C2) follows the same reasoning as that of (C1).

- **Case** $\tau = \tau_1 \rightarrow \tau_2$

Our goal is

$$\forall v_1, v_2, \rho, i. i \notin \mathbf{FV}(\tau_1 \rightarrow \tau_2) \quad (\text{H})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \equiv (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^{\rho \setminus i} \quad (\text{C})$$

And the IH in τ_1 and τ_2 are of the form:

$$\forall v_{11}, v_{12}, \rho_1, i_1. i_1 \notin \mathbf{FV}(\tau_1) \quad (\text{P1})$$

$$\Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\tau_1]^{\rho_1} \equiv (v_{11}, v_{12}) \in \mathcal{V}[\tau_1]^{\rho_1 \setminus i_1} \quad (\text{IH1})$$

$$\forall v_{21}, v_{22}, \rho_2, i_2. i_2 \notin \mathbf{FV}(\tau_2) \quad (\text{P2})$$

$$\Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\tau_2]^{\rho_2} \equiv (v_{21}, v_{22}) \in \mathcal{V}[\tau_2]^{\rho_2 \setminus i_2} \quad (\text{IH2})$$

Now, to prove (C) we can decompose it in both directions of the equivalence as follows:

$$\forall v_1, v_2, \rho, i. i \notin \mathbf{FV}(\tau_1 \rightarrow \tau_2) \quad (\text{H1.1})$$

$$\wedge (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \quad (\text{H1.2})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^{\rho \setminus i} \quad (\text{C1})$$

$$\forall v_1, v_2, \rho, i. i \notin \mathbf{FV}(\tau_1 \rightarrow \tau_2) \quad (\text{H2.1})$$

$$\wedge (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^{\rho \setminus i} \quad (\text{H2.2})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \quad (\text{C2})$$

Let's start by proving (C1). Expanding (H1.2) we have:

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \\
& \equiv \langle \text{By cases in } v_1, v_2 \rangle \\
& \quad (\langle \lambda x.e_1 \mid \gamma_1 \rangle, \langle \lambda x.e_2 \mid \gamma_2 \rangle) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (_ \rightarrow _) \rangle \\
& \quad \forall u_{11}, u_{12}. (u_{11}, u_{12}) \in \mathcal{V}[\tau_1]^\rho \\
& \quad \Rightarrow (\gamma_1 \vdash \lambda x.e_1 @ u_{11}, \gamma_2 \vdash \lambda x.e_2 @ u_{12}) \in \mathcal{E}[\tau_2]^\rho_\Gamma \\
& \equiv \langle \text{By def of } \mathcal{E}[_]^\rho_\Gamma \rangle \\
& \quad \forall u_{11}, u_{12}, u_{11}^*, u_{12}^*. (u_{11}, u_{12}) \in \mathcal{V}[\tau_1]^\rho \tag{A1.1} \\
& \quad \wedge \gamma_1 \vdash \lambda x.e_1 @ u_{11} \Downarrow u_{11}^* \tag{A1.2} \\
& \quad \wedge \gamma_2 \vdash \lambda x.e_2 @ u_{12} \Downarrow u_{12}^* \tag{A1.3} \\
& \quad \Rightarrow (u_{11}^*, u_{12}^*) \in \mathcal{V}[\tau_2]^\rho \tag{H'1.2}
\end{aligned}$$

Similarly, we can inspect the goal (C1) as follows:

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^{\rho \setminus i} \\
& \equiv \langle \text{By cases in } v_1, v_2 \rangle \\
& \quad (\langle \lambda x.e_1 \mid \gamma_1 \rangle, \langle \lambda x.e_2 \mid \gamma_2 \rangle) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^{\rho \setminus i} \\
& \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (_ \rightarrow _) \rangle \\
& \quad \forall u_{21}, u_{22}. (u_{21}, u_{22}) \in \mathcal{V}[\tau_1]^{\rho \setminus i} \\
& \quad \Rightarrow (\gamma_1 \vdash \lambda x.e_1 @ u_{21}, \gamma_2 \vdash \lambda x.e_2 @ u_{22}) \in \mathcal{E}[\tau_2]^{\rho \setminus i}_\Gamma \\
& \equiv \langle \text{By def of } \mathcal{E}[_]^\rho_\Gamma \rangle \\
& \quad \forall u_{21}, u_{22}, u_{21}^*, u_{22}^*. (u_{21}, u_{22}) \in \mathcal{V}[\tau_1]^{\rho \setminus i} \tag{H1.3} \\
& \quad \wedge \gamma_1 \vdash \lambda x.e_1 @ u_{21} \Downarrow u_{21}^* \tag{H1.4} \\
& \quad \wedge \gamma_2 \vdash \lambda x.e_2 @ u_{22} \Downarrow u_{22}^* \tag{H1.5} \\
& \quad \Rightarrow (u_{21}^*, u_{22}^*) \in \mathcal{V}[\tau_2]^{\rho \setminus i} \tag{C1'}
\end{aligned}$$

Let $u_{11} = u_{21}$, $u_{12} = u_{22}$, $u_{11}^* = u_{21}^*$, and $u_{12}^* = u_{22}^*$. Observe that we can simplify (H'1.2) since (H1.4) and (H1.5) satisfy the conditions (A1.2), and (A1.3), respectively. With this, our objective is of the form:

$$\forall u_{11}, u_{12}, u_{11}^*, u_{12}^*, \rho, i. i \notin \mathbf{FV}(\tau_1 \rightarrow \tau_2) \quad (\text{H1.1})$$

$$\wedge ((u_{11}, u_{12}) \in \mathcal{V}[\tau_1]^\rho \Rightarrow (u_{11}^*, u_{12}^*) \in \mathcal{V}[\tau_2]^\rho) \quad (\text{H'1.2})$$

$$\wedge (u_{11}, u_{12}) \in \mathcal{V}[\tau_1]^{\rho \setminus i} \quad (\text{H1.3})$$

$$\wedge \gamma_1 \vdash \lambda x. e_1 @ u_{11} \Downarrow u_{11}^* \quad (\text{H1.4})$$

$$\wedge \gamma_2 \vdash \lambda x. e_2 @ u_{12} \Downarrow u_{12}^* \quad (\text{H1.5})$$

$$\Rightarrow (u_{11}^*, u_{12}^*) \in \mathcal{V}[\tau_2]^{\rho \setminus i} \quad (\text{C1'})$$

Now, recall the IH in τ_1 (IH1) and τ_2 (IH2). Let $i_1 = i_2 = i$, $\rho_1 = \rho_2 = \rho$, $v_{11} = u_{11}$, $v_{12} = u_{12}$, $v_{21} = u_{11}^*$, and $v_{22} = u_{12}^*$. By (H1.1) we know that $i \notin \mathbf{FV}(\tau_1 \rightarrow \tau_2)$, then it must be the case that $i \notin \mathbf{FV}(\tau_1)$ (proving (P1)), and $i \notin \mathbf{FV}(\tau_2)$ (proving (P2)). Then, from (IH1) and (IH2) we conclude:

$$(u_{11}, u_{12}) \in \mathcal{V}[\tau_1]^\rho \equiv (u_{11}, u_{12}) \in \mathcal{V}[\tau_1]^{\rho \setminus i} \quad (\text{IH1'})$$

$$\wedge (u_{11}^*, u_{12}^*) \in \mathcal{V}[\tau_2]^\rho \equiv (u_{11}^*, u_{12}^*) \in \mathcal{V}[\tau_2]^{\rho \setminus i} \quad (\text{IH2'})$$

By (H1.3) and (IH'1) we get $(u_{11}, u_{12}) \in \mathcal{V}[\tau_1]^\rho$ which can be applied to (H'1.2) to obtain $(u_{11}^*, u_{12}^*) \in \mathcal{V}[\tau_2]^\rho$, then, by (IH'2) we get $(u_{11}^*, u_{12}^*) \in \mathcal{V}[\tau_2]^{\rho \setminus i}$ which is exactly our goal (C1').

Proving (C2) follows the same reasoning as (C1).

- **Case** $\tau = \text{Rel } d \sigma$ is proven by Lemma B.6

□

Lemma B.8. *If $(v_1, v_2) \in \mathcal{V}[\tau]^\rho$, $i \notin \mathbf{FV}(\tau)$ then $(v_1, v_2) \in \mathcal{V}[\tau]^{\rho \setminus i}$.*

Proof. Follows from Lemma B.7

□

B.E.1 Fundamental lemma of logical relations

Lemma B.9. *Let a well typed expression $\Psi; \mathcal{C}; \Gamma \vdash e : \tau$ be given. For any ρ for which $\Psi \subseteq \text{dom}(\rho)$ and $\rho \models \mathcal{C}$; suppose γ_1, γ_2 are two substitutions for Γ such that $(\gamma_1, \gamma_2) \in \mathcal{S}[\Gamma]^\rho$, then we have $(\gamma_1 \vdash e, \gamma_2 \vdash e) \in \mathcal{E}[\tau]_\Gamma^\rho$. In other words:*

$$\forall \Psi, \mathcal{C}, \Gamma, e, \tau, \rho, \gamma_1, \gamma_2. \Psi; \mathcal{C}; \Gamma \vdash e : \tau \quad (\text{H1})$$

$$\wedge \Psi \subseteq \text{dom}(\rho) \wedge \rho \models \mathcal{C} \quad (\text{H2})$$

$$\wedge (\gamma_1, \gamma_2) \in \mathcal{S}[\Gamma]^\rho \quad (\text{H3})$$

$$\Rightarrow (\gamma_1 \vdash e, \gamma_2 \vdash e) \in \mathcal{E}[\tau]_\Gamma^\rho \quad (\text{C})$$

Proof. By induction on the typing derivations of e

- **Case [T.VAR]:** $\Psi; \mathcal{C}; \Gamma \vdash x : \tau$

$$\begin{aligned}
\text{Goal} &: \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash x, \gamma_2 \vdash x) \in \mathcal{E}[\![\tau]\!]_{\Gamma}^{\rho} \\
&\equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
&\quad \gamma_1 \vdash x \Downarrow v_1 \wedge \tag{H4} \\
&\quad \gamma_2 \vdash x \Downarrow v_2 \tag{H5} \\
&\quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^{\rho} \tag{C'}
\end{aligned}$$

By **E.VAR** we know $\gamma_1 \vdash x \Downarrow \gamma_1(x)$, and $\gamma_2 \vdash x \Downarrow \gamma_2(x)$. Let $v_1 = \gamma_1(x)$ and $v_2 = \gamma_2(x)$; then we need to show that $(\gamma_1(x), \gamma_2(x)) \in \mathcal{V}[\![\tau]\!]$ which is proven by (H3) and definition of $\mathcal{S}[\![_]\!]$.

- **Case [T.NUM]:** $\Psi; \mathcal{C}; \Gamma \vdash n : \mathbb{R}$

$$\begin{aligned}
\text{Goal} &: \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash n, \gamma_2 \vdash n) \in \mathcal{E}[\![\mathbb{R}]\!]_{\Gamma}^{\rho} \\
&\equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
&\quad \gamma_1 \vdash n \Downarrow v_1 \wedge \tag{H4} \\
&\quad \gamma_2 \vdash n \Downarrow v_2 \tag{H5} \\
&\quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho} \tag{C'}
\end{aligned}$$

By **E.NUM** we have $v_1 \equiv v_2 \equiv n$, which satisfies the definition of $\mathcal{V}[\![_]\!]^{\rho}$ at \mathbb{R}

- **Case [T.ADD]:** $\Psi; \mathcal{C}; \Gamma \vdash (e_1 + e_2) : \mathbb{R}$

$$\begin{aligned}
\text{Goal} &: \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash (e_1 + e_2), \gamma_2 \vdash (e_1 + e_2)) \in \mathcal{E}[\![\mathbb{R}]\!]_{\Gamma}^{\rho} \\
&\equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
&\quad \gamma_1 \vdash (e_1 + e_2) \Downarrow v_1 \wedge \tag{H4} \\
&\quad \gamma_2 \vdash (e_1 + e_2) \Downarrow v_2 \tag{H5} \\
&\quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho} \tag{C'}
\end{aligned}$$

Consider the IH on both sub-expressions

$$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \mathbb{R}$$

$$\begin{aligned}
&\forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\
xz \text{ ""} &\Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\![\mathbb{R}]\!]_{\Gamma}^{\rho_1} \\
&\equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\
&\quad \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \tag{P1.1} \\
&\quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \tag{P1.2} \\
&\quad \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \tag{P1.3} \\
&\quad \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \tag{P1.4} \\
&\quad \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho_1} \tag{IH1.1}
\end{aligned}$$

$\Psi; \mathcal{C}; \Gamma \vdash e_2 : \mathbb{R}$

$$\begin{aligned}
& \forall \rho_2, \gamma_{21}, \gamma_{22}. \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \\
& \quad \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\![\mathbb{R}]\!]_{\Gamma}^{\rho_2} \\
& \equiv \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\
& \quad \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \tag{P2.1} \\
& \quad \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \tag{P2.2} \\
& \quad \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \tag{P2.3} \\
& \quad \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \tag{P2.4} \\
& \quad \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho_2} \tag{IH2.1}
\end{aligned}$$

Let $\rho_1 = \rho_2 = \rho$ (F1), then we can prove (P1.1) and (P2.1) by (H2).

Let $\gamma_{11} = \gamma_{21} = \gamma_1$ (F1.1) and $\gamma_{12} = \gamma_{22} = \gamma_2$ (F2.1). Then, by (H3), (F1), (F1.1), and (F2.1) we can prove (P1.2) and (P2.2), respectively.

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\begin{aligned}
& \gamma_1 \vdash (e_1 + e_2) \Downarrow v_1 \\
& \quad \Rightarrow \gamma_1 \vdash e_1 \Downarrow v'_{11} \wedge \gamma_1 \vdash e_2 \Downarrow v'_{21} \wedge v_1 = v'_{11} + v'_{21} \tag{F1.2} \\
& \gamma_2 \vdash (e_1 + e_2) \Downarrow v_2 \\
& \quad \Rightarrow \gamma_2 \vdash e_1 \Downarrow v'_{12} \wedge \gamma_2 \vdash e_2 \Downarrow v'_{22} \wedge v_2 = v'_{12} + v'_{22} \tag{F2.2}
\end{aligned}$$

Let $v_{11} = v'_{11}$ (F1.3) and $v_{12} = v'_{12}$ (F1.4), then (P1.3) and (P1.4) are proven by (F1.1-4) and F(2.1-2). Similarly, making $v_{21} = v'_{21}$ (F1.3) and $v_{22} = v'_{22}$ (F2.4), we can prove (P2.3) and (P2.4) by (F1.1-2) and F(2.1-4).

Having fulfilled the preconditions for (IH1.1) and (IH2.1) we can conclude that $(v_{11}, v_{12}) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho}$ and $(v_{21}, v_{22}) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho}$, which, by definition of $\mathcal{V}[\![_]\!]^{\rho}$ at \mathbb{R} , implies that $v_{11} \equiv v_{12}$ (F1.5) and $v_{21} \equiv v_{22}$ (F2.5).

Recall our objective (C')

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho} \\
& \equiv \langle \text{By (F1.2-4) and (F2.2-4)} \rangle \\
& \quad ((v_{11} + v_{21}), (v_{12} + v_{22})) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho} \\
& \equiv \langle \text{By def of } \mathcal{V}[\![_]\!]^{\rho} \text{ at } \mathbb{R} \rangle \\
& \quad (v_{11} + v_{21}) \equiv (v_{12} + v_{22}) \\
& \equiv \langle \text{By (F1.5) and (F2.5)} \rangle \\
& \quad (v_{11} + v_{21}) \equiv (v_{11} + v_{21})
\end{aligned}$$

Thus proving (C') and consequently (C).

- **Case [T.PAIR]:** $\Psi; \mathcal{C}; \Gamma \vdash (e_1, e_2) : (\tau_1 \times \tau_2)$

$$\begin{aligned}
\text{Goal} : & \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash (e_1, e_2), \gamma_2 \vdash (e_1, e_2)) \in \mathcal{E}[\![\tau_1 \times \tau_2]\!]_{\Gamma}^{\rho} \\
\equiv & \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \gamma_1 \vdash (e_1, e_2) \Downarrow v_1 \wedge & \text{(H4)} \\
& \gamma_2 \vdash (e_1, e_2) \Downarrow v_2 & \text{(H5)} \\
& \Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]^{\rho} & \text{(C')}
\end{aligned}$$

Similar to the previous case, we consider the IH on both sub-expressions

$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \tau_1$

$$\begin{aligned}
& \forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\
& \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\![\tau_1]\!]_{\Gamma}^{\rho_1} \\
\equiv & \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\
& \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} & \text{(P1.1)} \\
& \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} & \text{(P1.2)} \\
& \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} & \text{(P1.3)} \\
& \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} & \text{(P1.4)} \\
& \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\tau_1]\!]^{\rho_1} & \text{(IH1.1)}
\end{aligned}$$

$\Psi; \mathcal{C}; \Gamma \vdash e_2 : \tau_2$

$$\begin{aligned}
& \forall \rho_2, \gamma_{21}, \gamma_{22}. \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \\
& \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\![\tau_2]\!]_{\Gamma}^{\rho_2} \\
\equiv & \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\
& \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} & \text{(P2.1)} \\
& \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} & \text{(P2.2)} \\
& \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} & \text{(P2.3)} \\
& \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} & \text{(P2.4)} \\
& \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\tau_2]\!]^{\rho_2} & \text{(IH2.1)}
\end{aligned}$$

We take $\rho_1 = \rho_2 = \rho$ (F1) and prove (P1.1) and (P2.1) by (H2).

With $\gamma_{11} = \gamma_{21} = \gamma_1$ (F1.1), $\gamma_{12} = \gamma_{22} = \gamma_2$ (F2.1), (H3), and (F1), we can prove (P1.2) and (P2.2).

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\begin{aligned}
& \gamma_1 \vdash (e_1, e_2) \Downarrow v_1 \\
& \Rightarrow \gamma_1 \vdash e_1 \Downarrow v'_{11} \wedge \gamma_1 \vdash e_2 \Downarrow v'_{21} \wedge v_1 = (v'_{11}, v'_{21}) & \text{(F1.2)}
\end{aligned}$$

$$\begin{aligned}
& \gamma_2 \vdash (e_1, e_2) \Downarrow v_2 \\
& \Rightarrow \gamma_2 \vdash e_1 \Downarrow v'_{12} \wedge \gamma_2 \vdash e_2 \Downarrow v'_{22} \wedge v_2 = (v'_{12}, v'_{22}) & \text{(F2.2)}
\end{aligned}$$

Let $v_{11} = v'_{11}$ (F1.3) and $v_{12} = v'_{12}$ (F1.4), then (P1.3) and (P1.4) are proven by (F1.1-4) and F(2.1-2). Similarly, making $v_{21} = v'_{21}$ (F1.3) and $v_{22} = v'_{22}$ (F2.4), we can prove (P2.3) and (P2.4) by (F1.1-2) and F(2.1-4).

Having fulfilled the preconditions for (IH1.1) and (IH2.1) we can conclude that $(v_{11}, v_{12}) \in \mathcal{V}[\tau_1]^\rho$ (F1.5) and $(v_{21}, v_{22}) \in \mathcal{V}[\tau_2]^\rho$ (F2.5).

Recall our objective (C')

$$\begin{aligned}
 & (v_1, v_2) \in \mathcal{V}[\tau_1 \times \tau_2]^\rho \\
 & \equiv \langle \text{By (F1.2-4) and (F2.2-4)} \rangle \\
 & ((v_{11}, v_{21}), (v_{12}, v_{22})) \in \mathcal{V}[\tau_1 \times \tau_2]^\rho \\
 & \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (_ \times _) \rangle \\
 & (v_{11}, v_{12}) \in \mathcal{V}[\tau_1]^\rho \wedge (v_{21}, v_{22}) \in \mathcal{V}[\tau_2]^\rho
 \end{aligned}$$

Which are proven by (F1.5) and (F2.5).

- **Case [T.FST]:** $\Psi; \mathcal{C}; \Gamma \vdash \mathbf{fst} \ e' : \tau_1$

$$\begin{aligned}
 \text{Goal : } & \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash \mathbf{fst} \ e', \gamma_2 \vdash \mathbf{fst} \ e') \in \mathcal{E}[\tau_1]^\rho_\Gamma \\
 \equiv & \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
 & \gamma_1 \vdash \mathbf{fst} \ e' \Downarrow v_1 \wedge & \text{(H4)} \\
 & \gamma_2 \vdash \mathbf{fst} \ e' \Downarrow v_2 & \text{(H5)} \\
 \Rightarrow & (v_1, v_2) \in \mathcal{V}[\tau_1]^\rho & \text{(C')}
 \end{aligned}$$

Consider the IH on $\Psi; \mathcal{C}; \Gamma \vdash e' : \tau_1 \times \tau_2$

$$\begin{aligned}
 & \forall \rho', \gamma'_1, \gamma'_2. \Psi \subseteq \text{dom}(\rho') \wedge \rho' \models \mathcal{C} \wedge (\gamma'_1, \gamma'_2) \in \mathcal{S}[\Gamma]^\rho \\
 & \Rightarrow (\gamma'_1 \vdash e', \gamma'_2 \vdash e') \in \mathcal{E}[\tau_1 \times \tau_2]^\rho_\Gamma \\
 \equiv & \forall \rho', \gamma'_1, \gamma'_2, v'_1, v'_2. \\
 & \Psi \subseteq \text{dom}(\rho') \wedge \rho' \models \mathcal{C} & \text{(P1)} \\
 & \wedge (\gamma'_1, \gamma'_2) \in \mathcal{S}[\Gamma]^\rho & \text{(P2)} \\
 & \wedge \gamma'_1 \vdash e' \Downarrow v'_1 & \text{(P3)} \\
 & \wedge \gamma'_2 \vdash e' \Downarrow v'_2 & \text{(P4)} \\
 \Rightarrow & (v'_1, v'_2) \in \mathcal{V}[\tau_1 \times \tau_2]^\rho & \text{(IH)}
 \end{aligned}$$

With $\rho' = \rho$, $\gamma'_1 = \gamma_1$, and $\gamma'_2 = \gamma_2$, preconditions (P1) and (P2) are proven by (H2) and (H3), respectively.

By (H4), (H5), and cases on the evaluation relation $(_ \Downarrow _)$:

$$\gamma_1 \vdash \mathbf{fst} \ e' \Downarrow v_1 \Rightarrow \gamma_1 \vdash e' \Downarrow (v_1^*, x) \wedge v_1 = v_1^* \quad \text{(F1)}$$

$$\gamma_2 \vdash \mathbf{fst} \ e' \Downarrow v_2 \Rightarrow \gamma_2 \vdash e' \Downarrow (v_2^*, y) \wedge v_2 = v_2^* \quad \text{(F2)}$$

With $v'_1 = (v_1^*, x)$ (F3) and $v'_2 = (v_2^*, y)$ (F4) preconditions (P3) and (P4) are proven by (F1-4). Having (P1-4) fulfilled we have:

$$\begin{aligned}
 & (v'_1, v'_2) \in \mathcal{V}[\tau_1 \times \tau_2]^\rho \\
 & \equiv \langle \text{By (F3) and (F4)} \rangle \\
 & ((v_1^*, x), (v_2^*, y)) \in \mathcal{V}[\tau_1 \times \tau_2]^\rho \\
 & \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (_ \times _) \rangle \\
 & (v_1^*, v_2^*) \in \mathcal{V}[\tau_1]^\rho \wedge (x, y) \in \mathcal{V}[\tau_2]^\rho \\
 & \equiv \langle \text{By simplification and (F1), (F2)} \rangle \\
 & (v_1, v_2) \in \mathcal{V}[\tau_1]^\rho
 \end{aligned}$$

Which is exactly our goal (C').

- **Case [T.SND]:** $\Psi; \mathcal{C}; \Gamma \vdash \mathbf{snd} \ e' : \tau_2$
Similarly as the case of T.FST
- **Case [T.LAM]:** $\Psi; \mathcal{C}; \Gamma \vdash \lambda x. e' : \tau_1 \rightarrow \tau_2$

$$\begin{aligned}
 \text{Goal : } & \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash \lambda x. e', \gamma_2 \vdash \lambda x. e') \in \mathcal{E}[\tau_1 \rightarrow \tau_2]^\rho_\Gamma \\
 \equiv & \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
 & \gamma_1 \vdash \lambda x. e' \Downarrow v_1 \wedge & \text{(H4)} \\
 & \gamma_2 \vdash \lambda x. e' \Downarrow v_2 & \text{(H5)} \\
 \Rightarrow & (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho & \text{(C')}
 \end{aligned}$$

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\gamma_1 \vdash \lambda x. e' \Downarrow v_1 \Rightarrow v_1 = \langle \lambda x. e' \mid \gamma_1 \rangle \quad \text{(F1)}$$

$$\gamma_2 \vdash \lambda x. e' \Downarrow v_2 \Rightarrow v_2 = \langle \lambda x. e' \mid \gamma_2 \rangle \quad \text{(F2)}$$

Then, we can rewrite our objective (C') as follows:

$$\begin{aligned}
 & (v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \\
 & \equiv \langle \text{By (F1), (F2)} \rangle \\
 & (\langle \lambda x. e' \mid \gamma_1 \rangle, \langle \lambda x. e' \mid \gamma_2 \rangle) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \\
 & \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (_ \rightarrow _) \rangle \\
 & \forall v'_1, v'_2. (v'_1, v'_2) \in \mathcal{V}[\tau_1]^\rho \Rightarrow (\gamma_1 \vdash \lambda x. e' @ v'_1, \gamma_2 \vdash \lambda x. e' @ v'_2) \in \mathcal{E}[\tau_2]^\rho_\Gamma \\
 & \equiv \langle \text{By def of } \mathcal{E}[_]^\rho \rangle \\
 & \forall v'_1, v'_2, v_{13}, v_{23}. (v'_1, v'_2) \in \mathcal{V}[\tau_1]^\rho & \text{(H6)} \\
 & \wedge \gamma_1 \vdash \lambda x. e' @ v'_1 \Downarrow v_{13} & \text{(H7)} \\
 & \wedge \gamma_2 \vdash \lambda x. e' @ v'_2 \Downarrow v_{23} & \text{(H8)} \\
 \Rightarrow & (v_{13}, v_{23}) \in \mathcal{V}[\tau_2]^\rho & \text{(C'')}
 \end{aligned}$$

By (H7), (H8), (F1), (F2), and cases on $(_ \Downarrow _)$ we know that:

$$\begin{aligned} \gamma_1 \vdash \lambda x.e' @ v'_1 \Downarrow v_{13} &\Rightarrow \gamma_1 \vdash \lambda x.e' \Downarrow \langle \lambda x.e' \mid \gamma_1 \rangle \wedge \gamma_1 \vdash v'_1 \Downarrow v'_1 \\ &\wedge \gamma_1[x := v'_1] \vdash e' \Downarrow v_{13} \end{aligned} \quad (\text{F3})$$

$$\begin{aligned} \gamma_2 \vdash \lambda x.e' @ v'_2 \Downarrow v_{23} &\Rightarrow \gamma_2 \vdash \lambda x.e' \Downarrow \langle \lambda x.e' \mid \gamma_2 \rangle \wedge \gamma_2 \vdash v'_2 \Downarrow v'_2 \\ &\wedge \gamma_2[x := v'_2] \vdash e' \Downarrow v_{23} \end{aligned} \quad (\text{F4})$$

Now consider the IH on $\Psi; \mathcal{C}; \Gamma, x : \tau_1 \vdash e' : \tau_2$

$$\begin{aligned} &\forall \rho', \gamma'_1, \gamma'_2. \\ &\Psi \subseteq \text{dom}(\rho') \wedge \rho' \models \mathcal{C} \end{aligned} \quad (\text{P1})$$

$$\wedge (\gamma'_1, \gamma'_2) \in \mathcal{S}[\![\Gamma, x : \tau_1]\!]^{\rho'} \quad (\text{P2})$$

$$\Rightarrow (\gamma'_1 \vdash e', \gamma'_2 \vdash e') \in \mathcal{E}[\![\tau_2]\!]_{\Gamma, x: \tau_1}^{\rho'} \quad (\text{IH})$$

Let $\rho' = \rho$ such that (P1) is proven by (H2).

Take $\gamma'_1 = \gamma_1[x := v'_1]$ and $\gamma'_2 = \gamma_2[x := v'_2]$. Since $(\gamma_1, \gamma_2) \in \mathcal{S}[\![\Gamma]\!]^\rho$ (by (H3)), and $(v'_1, v'_2) \in \mathcal{V}[\![\tau_1]\!]^\rho$ (by (H6)), then it follows that $(\gamma'_1, \gamma'_2) \in \mathcal{S}[\![\Gamma, x : \tau_1]\!]^\rho$ (by definition of $\mathcal{S}[\![__]^\rho$), which proves (P2). Then we can apply (IH) and obtain:

$$\begin{aligned} &(\gamma_1[x := v'_1] \vdash e', \gamma_2[x := v'_2] \vdash e') \in \mathcal{E}[\![\tau_2]\!]_{\Gamma, x: \tau_1}^\rho \\ &\equiv \langle \text{By def of } \mathcal{E}[\![__]^\rho \rangle \\ &\quad \forall v_1^*, v_2^*. \gamma_1[x := v'_1] \vdash e' \Downarrow v_1^* \wedge \gamma_2[x := v'_2] \vdash e' \Downarrow v_2^* \\ &\quad \Rightarrow (v_1^*, v_2^*) \in \mathcal{V}[\![\tau_2]\!]^\rho \end{aligned} \quad (\text{IH}')$$

If we take $v_1^* = v_{13}$ and $v_2^* = v_{23}$, then, by (F3) and (F4) we know that $\gamma_1[x := v'_1] \vdash e' \Downarrow v_1^*$ and $\gamma_2[x := v'_2] \vdash e' \Downarrow v_2^*$ allowing us to conclude (by (IH')) that $(v_1^*, v_2^*) \in \mathcal{V}[\![\tau_2]\!]^\rho$, which can be rewritten as $(v_{13}, v_{23}) \in \mathcal{V}[\![\tau_2]\!]^\rho$ proving (C'')—consequently (C') and (C) hold.

- **Case [T.App]:** $\Psi; \mathcal{C}; \Gamma \vdash e_1 @ e_2 : \tau_2$

$$\begin{aligned} \text{Goal : } &\forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash e_1 @ e_2, \gamma_2 \vdash e_1 @ e_2) \in \mathcal{E}[\![\tau_2]\!]_\Gamma^\rho \\ &\equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \end{aligned} \quad (\text{H4})$$

$$\begin{aligned} &\gamma_1 \vdash e_1 @ e_2 \Downarrow v_1 \wedge \\ &\gamma_2 \vdash e_1 @ e_2 \Downarrow v_2 \end{aligned} \quad (\text{H5})$$

$$\Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau_2]\!]^\rho \quad (\text{C}')$$

Consider the IH on both sub-expressions

$$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2$$

$$\begin{aligned} & \forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\ & \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\![\tau_1 \rightarrow \tau_2]\!]_{\Gamma}^{\rho_1} \\ \equiv & \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\ & \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} & \text{(P1.1)} \\ & \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} & \text{(P1.2)} \\ & \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} & \text{(P1.3)} \\ & \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} & \text{(P1.4)} \\ & \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!]^{\rho_1} & \text{(IH1.1)} \end{aligned}$$

$$\Psi; \mathcal{C}; \Gamma \vdash e_2 : \tau_1$$

$$\begin{aligned} & \forall \rho_2, \gamma_{21}, \gamma_{22}. \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \\ & \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\![\tau_1]\!]_{\Gamma}^{\rho_2} \\ \equiv & \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\ & \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} & \text{(P2.1)} \\ & \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} & \text{(P2.2)} \\ & \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} & \text{(P2.3)} \\ & \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} & \text{(P2.4)} \\ & \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\tau_1]\!]^{\rho_2} & \text{(IH2.1)} \end{aligned}$$

Let $\rho_1 = \rho_2 = \rho$ (F1), then we can prove (P1.1) and (P2.1) by (H2).

With $\gamma_{11} = \gamma_{21} = \gamma_1$ (F1.1), $\gamma_{12} = \gamma_{22} = \gamma_2$ (F2.1), (H3), and (F1), we can prove (P1.2) and (P2.2).

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\begin{aligned} & \gamma_1 \vdash e_1 @ e_2 \Downarrow v_1 \\ & \Rightarrow \gamma_1 \vdash e_1 \Downarrow \langle \lambda x. e_{11} \mid \gamma'_1 \rangle \wedge \gamma_1 \vdash e_2 \Downarrow v'_{21} \wedge \gamma'_1[x := v'_{21}] \vdash e'_{11} \Downarrow v_1 & \text{(F1.2)} \end{aligned}$$

$$\begin{aligned} & \gamma_2 \vdash e_1 @ e_2 \Downarrow v_2 \\ & \Rightarrow \gamma_2 \vdash e_1 \Downarrow \langle \lambda x. e_{12} \mid \gamma'_2 \rangle \wedge \gamma_2 \vdash e_2 \Downarrow v'_{22} \wedge \gamma'_2[x := v'_{22}] \vdash e'_{21} \Downarrow v_2 & \text{(F2.2)} \end{aligned}$$

Let $v_{11} = \langle \lambda x. e_{11} \mid \gamma'_1 \rangle$ (F1.3) and $v_{12} = \langle \lambda x. e_{12} \mid \gamma'_2 \rangle$ (F1.4), then (P1.3) and (P1.4) are proven by (F1.1-4) and F(2.1-2). Similarly, making $v_{21} = v'_{21}$ (F1.3) and $v_{22} = v'_{22}$ (F2.4), we can prove (P2.3) and (P2.4) by (F1.1-2) and F(2.1-4).

Having fulfilled the preconditions for (IH1.1) and (IH2.1) we can conclude:

$$\begin{aligned}
& (v_{21}, v_{22}) \in \mathcal{V}[\tau_1]^\rho \\
& \equiv \langle \text{By (F2.3), (F2.4)} \rangle \\
& (v'_{21}, v'_{22}) \in \mathcal{V}[\tau_1]^\rho \tag{IH2.2}
\end{aligned}$$

$$\begin{aligned}
& (v_{11}, v_{12}) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \\
& \equiv \langle \text{By (F1.3), (F1.4)} \rangle \\
& (\langle \lambda x. e_{11} \mid \gamma'_1 \rangle, \langle \lambda x. e_{12} \mid \gamma'_2 \rangle) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (_ \rightarrow _) \rangle \\
& \forall v'_1, v'_2. (v'_1, v'_2) \in \mathcal{V}[\tau_1]^\rho \Rightarrow (\gamma'_1 \vdash e_{11} @ v'_1, \gamma'_2 \vdash e_{12} @ v'_2) \in \mathcal{E}[\tau_2]^\rho_\Gamma \\
& \equiv \langle \text{By def of } \mathcal{E}[_]^\rho \rangle \\
& \forall v'_1, v'_2, v_{13}, v_{23}. \\
& \quad (v'_1, v'_2) \in \mathcal{V}[\tau_1]^\rho \tag{P1.2.1} \\
& \quad \wedge \gamma'_1 \vdash e_{11} @ v'_1 \Downarrow v_{13} \tag{P1.2.2} \\
& \quad \wedge \gamma'_2 \vdash e_{12} @ v'_2 \Downarrow v_{23} \tag{P1.2.3} \\
& \quad \Rightarrow (v_{13}, v_{23}) \in \mathcal{V}[\tau_2]^\rho \tag{IH1.2}
\end{aligned}$$

Take $v'_1 = v'_{21}$, $v'_2 = v'_{22}$, $v_{13} = v_1$, and $v_{23} = v_2$; then (P1.2.1) is proven by (IH2.2), (P1.2.2) is proven by the fact that $\gamma'_1[x := v'_{21}] \vdash e'_{11} \Downarrow v_1$ (F1.2) and similarly, (P1.2.3) is proven by $\gamma'_2[x := v'_{22}] \vdash e'_{21} \Downarrow v_2$ (F2.2). Having proven (P1.2.1-3) we can conclude that $(v_{13}, v_{23}) \in \mathcal{V}[\tau_2]^\rho$ which is equivalent to $(v_1, v_2) \in \mathcal{V}[\tau_2]^\rho$ —corresponding to our objective (C').

- **Case [T.NIL]:** $\Psi, j; \mathcal{C} \wedge j = 0; \Gamma \vdash [] : \vec{\tau}_j$

$$\begin{aligned}
& \text{Goal : } \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash [], \gamma_2 \vdash []) \in \mathcal{E}[\vec{\tau}_j]^\rho_\Gamma \\
& \equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \quad \gamma_1 \vdash [] \Downarrow v_1 \wedge \tag{H4} \\
& \quad \gamma_2 \vdash [] \Downarrow v_2 \tag{H5} \\
& \quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\vec{\tau}_j]^\rho \tag{C'}
\end{aligned}$$

By **E.NIL** and cases on the evaluation relation ($_ \Downarrow _$) we know $\gamma_1 \vdash [] \Downarrow []$, and $\gamma_2 \vdash [] \Downarrow []$. Let $v_1 = v_2 = []$, then we need to show that $([], []) \in \mathcal{V}[\vec{\tau}_j]^\rho$. By (H2) we know that $\rho \models \mathcal{C} \wedge j = 0$, then $\llbracket j \rrbracket_{\rho,1} = 0$ (by definition of $(_ \models _)$ and $\llbracket _ \rrbracket_{\rho,1}$). With this, we can prove that $([], []) \in \mathcal{V}[\vec{\tau}_0]^\rho$ by definition of $\mathcal{V}[_]^\rho$ at $\vec{\tau}_0$.

- **Case [T.CONST]:** $\Psi, j; \mathcal{C} \wedge j = (l + 1); \Gamma \vdash e_1 :: e_2 : \vec{\tau}_j$

$$\begin{aligned}
\text{Goal} &: \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash e_1 :: e_2, \gamma_2 \vdash e_1 :: e_2) \in \mathcal{E}[\![\vec{\tau}_j]\!]_{\Gamma}^{\rho} \\
&\equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
&\quad \gamma_1 \vdash e_1 :: e_2 \Downarrow v_1 \wedge \tag{H4} \\
&\quad \gamma_2 \vdash e_1 :: e_2 \Downarrow v_2 \tag{H5} \\
&\quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\![\vec{\tau}_j]\!]^{\rho} \tag{C'}
\end{aligned}$$

Similar to the case of pairs, we consider the IH on both sub-expressions

$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \tau$

$$\begin{aligned}
&\forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\
&\quad \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\![\tau]\!]_{\Gamma}^{\rho_1} \\
&\equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\
&\quad \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \tag{P1.1} \\
&\quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \tag{P1.2} \\
&\quad \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \tag{P1.3} \\
&\quad \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \tag{P1.4} \\
&\quad \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\tau]\!]^{\rho_1} \tag{IH1.1}
\end{aligned}$$

$\Psi; \mathcal{C}; \Gamma \vdash e_2 : \vec{\tau}_l$

$$\begin{aligned}
&\forall \rho_2, \gamma_{21}, \gamma_{22}. \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \\
&\quad \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\![\vec{\tau}_l]\!]_{\Gamma}^{\rho_2} \\
&\equiv \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\
&\quad \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \tag{P2.1} \\
&\quad \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \tag{P2.2} \\
&\quad \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \tag{P2.3} \\
&\quad \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \tag{P2.4} \\
&\quad \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\vec{\tau}_l]\!]^{\rho_2} \tag{IH2.1}
\end{aligned}$$

We take $\rho_1 = \rho_2 = \rho$ (F1) and prove (P1.1) and (P2.1) by (H2).

With $\gamma_{11} = \gamma_{21} = \gamma_1$ (F1.1), $\gamma_{12} = \gamma_{22} = \gamma_2$ (F2.1), (H3), and (F1), we can prove (P1.2) and (P2.2).

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\begin{aligned}
&\gamma_1 \vdash e_1 :: e_2 \Downarrow v_1 \\
&\quad \Rightarrow \gamma_1 \vdash e_1 \Downarrow v'_{11} \wedge \gamma_1 \vdash e_2 \Downarrow v'_{21} \wedge v_1 = v'_{11} :: v'_{21} \tag{F1.2}
\end{aligned}$$

$$\begin{aligned}
&\gamma_2 \vdash e_1 :: e_2 \Downarrow v_2 \\
&\quad \Rightarrow \gamma_2 \vdash e_1 \Downarrow v'_{12} \wedge \gamma_2 \vdash e_2 \Downarrow v'_{22} \wedge v_2 = v'_{12} :: v'_{22} \tag{F2.2}
\end{aligned}$$

Let $v_{11} = v'_{11}$ (F1.3) and $v_{12} = v'_{12}$ (F1.4), then (P1.3) and (P1.4) are proven by (F1.1-4) and F(2.1-2). Similarly, making $v_{21} = v'_{21}$ (F1.3) and $v_{22} = v'_{22}$ (F2.4), we can prove (P2.3) and (P2.4) by (F1.1-2) and F(2.1-4).

Having fulfilled the preconditions for (IH1.1) and (IH2.1) we can conclude that $(v_{11}, v_{12}) \in \mathcal{V}[\tau]^\rho$ (F1.5) and $(v_{21}, v_{22}) \in \mathcal{V}[\vec{\tau}_l]^\rho$ (F2.5).

By (H2) we know that $\rho \models \mathcal{C} \wedge j = (l + 1)$, then, by definition of $(_ \models _)$ and $\llbracket j \rrbracket_\rho$, we know that $\llbracket j \rrbracket_\rho \equiv \llbracket l \rrbracket_\rho + 1$. Let $\llbracket l \rrbracket_\rho = m$ (with $m \in \mathbb{N}$), then $\llbracket j \rrbracket_\rho \equiv m + 1$ (F2).

Recall our objective (C')

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\vec{\tau}_j]^\rho \\
& \equiv \langle \text{By (F1.2-4) and (F2.2-4)} \rangle \\
& (v_{11} :: v_{21}, v_{12} :: v_{22}) \in \mathcal{V}[\vec{\tau}_j]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at } (\vec{\tau}_l) \text{ with } \llbracket j \rrbracket_\rho \neq 0 \rangle \\
& (v_{11}, v_{12}) \in \mathcal{V}[\tau]^\rho \wedge (v_{21}, v_{22}) \in \mathcal{V}[\vec{\tau}_{(\llbracket j \rrbracket_\rho - 1)}]^\rho \\
& \equiv \langle \text{By (F2) and subtraction} \rangle \\
& (v_{11}, v_{12}) \in \mathcal{V}[\tau]^\rho \wedge (v_{21}, v_{22}) \in \mathcal{V}[\vec{\tau}_m]^\rho
\end{aligned}$$

We can prove $(v_{11}, v_{12}) \in \mathcal{V}[\tau]^\rho$ directly by (F1.5). To prove $(v_{21}, v_{22}) \in \mathcal{V}[\vec{\tau}_m]^\rho$ we use (F2.5), the assignment $\llbracket l \rrbracket_\rho = m$, and the fact that for any two values $(u_1, u_2) \in \mathcal{V}[\vec{\tau}_l]^\rho$ then it must be the case that $(u_1, u_2) \in \mathcal{V}[\vec{\tau}_{\llbracket l \rrbracket_\rho}]^\rho$ —this can be proven similarly to Lemma B.5.

- **Case [T.CASE]:** $\Psi; \mathcal{C}; \Gamma \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} : \tau$

Goal : $\forall \rho, \gamma_1, \gamma_2.$

$$\begin{aligned}
& (\gamma_1 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \\
& , \gamma_1 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\}) \in \mathcal{E}[\tau]^\rho \\
& \equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \gamma_1 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \Downarrow v_1 \wedge \tag{H4} \\
& \gamma_2 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \Downarrow v_2 \tag{H5} \\
& \Rightarrow (v_1, v_2) \in \mathcal{V}[\tau]^\rho \tag{C'}
\end{aligned}$$

By cases on the typing derivation of $\Psi; \mathcal{C}; \Gamma \vdash e_1 : \vec{\tau}'_l$ we obtain:

- **Case [T.NIL]:**

$$e_1 : \Psi', j'; \mathcal{C}' \wedge j' = 0; \Gamma \vdash [_] : \vec{\tau}'_{j'}$$

In this case we have

$$\Psi \equiv \Psi', j' \quad (\text{D1})$$

$$\mathcal{C} \equiv \mathcal{C}' \wedge j' = 0 \quad (\text{D2})$$

$$l \equiv j' \quad (\text{D3})$$

$$e_1 \equiv [] \quad (\text{D4})$$

By (H4), (H5), (D4), and cases on the evaluation relation ($_ \Downarrow _$) we have:

$$\begin{aligned} \gamma_1 \vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[] \cdot e_2\} \{(x :: xs) \cdot e_3\} \Downarrow v_1 \\ \Rightarrow \gamma_1 \vdash e_1 \Downarrow [] \wedge \gamma_1 \vdash e_2 \Downarrow v_1 \end{aligned} \quad (\text{F1})$$

$$\begin{aligned} \gamma_2 \vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[] \cdot e_2\} \{(x :: xs) \cdot e_3\} \Downarrow v_2 \\ \Rightarrow \gamma_2 \vdash e_1 \Downarrow [] \wedge \gamma_2 \vdash e_2 \Downarrow v_2 \end{aligned} \quad (\text{F2})$$

Now, let's consider the IH on e_2 :

$$\Psi; \mathcal{C} \wedge l = 0; \Gamma \vdash e_2 : \tau$$

$$\forall \rho_2, \gamma_{21}, \gamma_{22}.$$

$$\Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge l = 0$$

$$\wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2}$$

$$\Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\![\tau]\!]_{\Gamma}^{\rho_2}$$

$$\equiv \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}.$$

$$\Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge l = 0 \quad (\text{P2.1})$$

$$\wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \quad (\text{P2.2})$$

$$\wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \quad (\text{P2.3})$$

$$\wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \quad (\text{P2.4})$$

$$\Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\tau]\!]^{\rho_2} \quad (\text{IH2.1})$$

Let $\rho_2 = \rho$ (F3), $\gamma_{21} = \gamma_1 \wedge \gamma_{22} = \gamma_2$ (F4); then (P2.1) is proven by (H2) given that $l \equiv j'$ (D3). Similarly, premise (P2.2) is proven by (F1-4) together with (H3).

Let $v_{21} = v_1$ and $v_{22} = v_2$, then we can prove (P2.3) and (P2.4) by (F1) and (F2), respectively. Having proven all the premises (P2.1-4) we obtain $(v_1, v_2) \in \mathcal{V}[\![\tau]\!]^{\rho}$, that is no less than our objective (C').

– **Case [T.CONs]:**

$$e_1 : \Psi', j'; \mathcal{C}' \wedge j' = (l' + 1); \Gamma \vdash e'_1 :: es'_1 : \vec{\tau'}_{j'}$$

In this case we have

$$\Psi \equiv \Psi', j' \quad (\text{D1})$$

$$\mathcal{C} \equiv \mathcal{C}' \wedge j' = (l' + 1) \quad (\text{D2})$$

$$l \equiv j' \quad (\text{D3})$$

$$e_1 \equiv e'_1 :: es'_1 \quad (\text{D4})$$

By (H4), (H5), (D4), and cases on the evaluation relation ($_ \Downarrow _$) we have:

$$\begin{aligned} \gamma_1 &\vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[\] . e_2\} \{(x :: xs). e_3\} \Downarrow v_1 \\ &\Rightarrow \gamma_1 \vdash e_1 \Downarrow v'_{11} :: vs'_{21} \\ &\quad \wedge \gamma_1[x := v'_{11}][xs := vs'_{21}] \vdash e_3 \Downarrow v_1 \end{aligned} \quad (\text{F1})$$

$$\begin{aligned} \gamma_2 &\vdash \mathbf{case} \ e_1 \ \mathbf{of} \ \{[\] . e_2\} \{(x :: xs). e_3\} \Downarrow v_2 \\ &\Rightarrow \gamma_2 \vdash e_1 \Downarrow v'_{12} :: vs'_{22} \\ &\quad \wedge \gamma_2[x := v'_{12}][xs := vs'_{22}] \vdash e_3 \Downarrow v_2 \end{aligned} \quad (\text{F2})$$

Lets consider the IH on e_1 :

$$\begin{aligned} &\forall \rho_1, \gamma_{11}, \gamma_{12}. (\Psi', j') \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C}' \wedge j' = (l' + 1) \\ &\quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\ &\quad \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\![\vec{\tau}'_{j'}]\!]_{\Gamma}^{\rho_1} \\ &\equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\ &\quad (\Psi', j') \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C}' \wedge j' = (l' + 1) \quad (\text{P1.1}) \\ &\quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \quad (\text{P1.2}) \\ &\quad \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \quad (\text{P1.3}) \\ &\quad \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \quad (\text{P1.4}) \\ &\quad \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\vec{\tau}'_{j'}]\!]^{\rho_1} \quad (\text{IH1.1}) \end{aligned}$$

Let $\rho_1 = \rho$ (F1.1), $\gamma_{11} = \gamma_1 \wedge \gamma_{12} = \gamma_2$ (F1.2); then (P1.1) is proven by (H2), (D1), and (D2). Similarly, premise (P1.2) is proven by (F1.1-2), (D1), (D2), together with (H3).

If we take $v_{11} = v'_{11} :: vs'_{21}$ (F1.3) and $v_{12} = v'_{12} :: vs'_{22}$ (F1.4) we can prove (P1.3) and (P1.4) by (F1) and (F2). Having proven all the premises (P1.1-4) we obtain:

$$\begin{aligned} &(v'_{11} :: vs'_{21}, v'_{12} :: vs'_{22}) \in \mathcal{V}[\![\vec{\tau}'_{j'}]\!]^{\rho} \\ &\equiv \langle \text{By def of } \mathcal{V}[\![_]\!]^{\rho} \text{ at } \vec{\tau}_l \text{ with } [\![l]\!]_{\rho,1} \neq 0 \rangle \\ &\quad (v'_{11}, v'_{12}) \in \mathcal{V}[\![\tau']]\!]^{\rho} \wedge (vs'_{21}, vs'_{22}) \in \mathcal{V}[\![\vec{\tau}'_{(j')_{\rho,1-1}}]\!]^{\rho} \\ &\equiv \langle \text{By (D2), (D3) and } \rho \models \mathcal{C}' \wedge j' = (l' + 1) \rangle \\ &\quad (v'_{11}, v'_{12}) \in \mathcal{V}[\![\tau']]\!]^{\rho} \wedge (vs'_{21}, vs'_{22}) \in \mathcal{V}[\![\vec{\tau}'_{(l')_{\rho,1+1-1}}]\!]^{\rho} \\ &\equiv \langle \text{By simplification} \rangle \\ &\quad (v'_{11}, v'_{12}) \in \mathcal{V}[\![\tau']]\!]^{\rho} \wedge (vs'_{21}, vs'_{22}) \in \mathcal{V}[\![\vec{\tau}'_{[l']_{\rho,1}}]\!]^{\rho} \quad (\text{IH1.2}) \end{aligned}$$

Now, lets consider the IH on e_3 :

$\Psi, j; \mathcal{C} \wedge l = (j + 1); \Gamma, x : \tau', xs : \vec{\tau}'_j \vdash e_3 : \tau$ with $j \notin \Psi.1$

$$\begin{aligned}
& \forall \rho_3, \gamma_{31}, \gamma_{32}. \\
& (\Psi, j) \subseteq \text{dom}(\rho_3) \wedge \rho_3 \models \mathcal{C} \wedge l = (j + 1) \\
& \wedge (\gamma_{31}, \gamma_{32}) \in \mathcal{S}[\Gamma, x : \tau', xs : \vec{\tau}'_j]^{\rho_3} \\
& \Rightarrow (\gamma_{31} \vdash e_3, \gamma_{32} \vdash e_3) \in \mathcal{E}[\tau]^{\rho_3}_{(\Gamma, x:\tau', xs:\vec{\tau}'_j)} \\
& \equiv \forall \rho_3, \gamma_{31}, \gamma_{32}, v_{31}, v_{32}. \\
& (\Psi, j) \subseteq \text{dom}(\rho_3) \wedge \rho_3 \models \mathcal{C} \wedge l = (j + 1) \tag{P3.1} \\
& \wedge (\gamma_{31}, \gamma_{32}) \in \mathcal{S}[\Gamma, x : \tau', xs : \vec{\tau}'_j]^{\rho_3} \tag{P3.2} \\
& \wedge \gamma_{31} \vdash e_3 \Downarrow v_{31} \tag{P3.3} \\
& \wedge \gamma_{32} \vdash e_3 \Downarrow v_{32} \tag{P3.4} \\
& \Rightarrow (v_{31}, v_{32}) \in \mathcal{V}[\tau]^{\rho_3} \tag{IH3.1}
\end{aligned}$$

$$\text{Let } \rho_3 = \rho[j := \llbracket l' \rrbracket_{\rho.1}] \tag{F3.1}$$

$$\gamma_{31} = \gamma_1[x := v'_{11}][xs := vs'_{21}] \tag{F3.2}$$

$$\gamma_{32} = \gamma_2[x := v'_{12}][xs := vs'_{22}] \tag{F3.3}$$

We can prove $(\Psi, j) \subseteq \text{dom}(\rho[j := \llbracket l' \rrbracket_{\rho.1}])$ by (H2) and definition of $(_ \subseteq _)$. Now, to see that $\rho_3 \models \mathcal{C} \wedge l = (j + 1)$ we need to unfold the definition of $(_ \models _)$ as follows:

$$\begin{aligned}
& \rho_3 \models \mathcal{C} \wedge l = (j + 1) \\
& \equiv \langle \text{By (F3)} \rangle \\
& \rho[j := \llbracket l' \rrbracket_{\rho.1}] \models \mathcal{C} \wedge l = (j + 1) \\
& \equiv \langle \text{By (D2)} \rangle \\
& \rho[j := \llbracket l' \rrbracket_{\rho.1}] \models \mathcal{C}' \wedge j' = (l' + 1) \wedge l = (j + 1) \\
& \equiv \langle \text{By (D3)} \rangle \\
& \rho[j := \llbracket l' \rrbracket_{\rho.1}] \models \mathcal{C}' \wedge j' = (l' + 1) \wedge j' = (j + 1) \\
& \equiv \langle \text{By def of } (_ \models _) \rangle \\
& \rho[j := \llbracket l' \rrbracket_{\rho.1}] \models \mathcal{C}' \tag{P3.1.1} \\
& \wedge \rho[j := \llbracket l' \rrbracket_{\rho.1}] \models j' = (l' + 1) \tag{P3.1.2} \\
& \wedge \rho[j := \llbracket l' \rrbracket_{\rho.1}] \models j' = (j + 1) \tag{P3.1.3}
\end{aligned}$$

Given that $\rho \models \mathcal{C}' \wedge j' = (l' + 1)$ (by (H2) and (D2)) then it must be the case that $\rho[j := \llbracket l' \rrbracket_{\rho.1}] \models \mathcal{C}' \wedge j' = (l' + 1)$, proving (P3.1.1) and (P3.1.2). From the latter, we have that:

$$\begin{aligned}
& \llbracket j' \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket l' + 1 \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \\
& \equiv \langle \text{By def of } \llbracket _ \rrbracket_{\rho} \rangle \\
& \llbracket j' \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket l' \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} + 1 \\
& \equiv \langle \text{Since } j \notin \text{vars}(l') \text{ because } j \notin \Psi.1 \rangle \\
& \llbracket j' \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket l' \rrbracket_{\rho.1} + 1
\end{aligned} \tag{F3.4}$$

To prove (P3.1.3) consider the definition of $(_ \models _)$

$$\begin{aligned}
& \rho[j := \llbracket l' \rrbracket_{\rho.1}] \models j' = (j + 1) \\
& \equiv \langle \text{By def of } (_ \models _) \rangle \\
& \llbracket j' \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket j + 1 \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \\
& \equiv \langle \text{By def of } \llbracket _ \rrbracket_{\rho} \rangle \\
& \llbracket j' \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket j \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} + 1 \\
& \equiv \langle \text{By def of } \llbracket _ \rrbracket_{\rho} \rangle \\
& \llbracket j' \rrbracket_{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket l' \rrbracket_{\rho.1} + 1
\end{aligned}$$

which is proven by (F3.4). With these we have proven all the conditions in (P3.1).

To prove $(\gamma_{31}, \gamma_{32}) \in \mathcal{S}[\Gamma, x : \tau', xs : \vec{\tau}'_j]^{\rho_3}$ (P3.2) we need to see that:

$$\begin{aligned}
& (\gamma_{31}(x), \gamma_{32}(x)) \in \mathcal{V}[\tau']^{\rho_3} \wedge (\gamma_{31}(xs), \gamma_{32}(xs)) \in \mathcal{V}[\vec{\tau}'_j]^{\rho_3} \\
& \equiv \langle \text{By (F3.1-3)} \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{V}[\tau']^{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \wedge (vs'_{21}, vs'_{22}) \in \mathcal{V}[\vec{\tau}'_j]^{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \\
& \equiv \langle \text{By Lemma B.8 knowing that } j \notin \Psi.1 \Rightarrow j \notin \mathbf{FV}(\tau') \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{V}[\tau']^{\rho} \wedge (vs'_{21}, vs'_{22}) \in \mathcal{V}[\vec{\tau}'_j]^{\rho[j:=\llbracket l' \rrbracket_{\rho.1}]} \\
& \equiv \langle \text{By interpreting } j \text{ and knowing that } j \notin \mathbf{FV}(\tau') \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{V}[\tau']^{\rho} \wedge (vs'_{21}, vs'_{22}) \in \mathcal{V}[\vec{\tau}'_{\llbracket l' \rrbracket_{\rho.1}}]^{\rho}
\end{aligned}$$

which holds directly by (IH1.2).

Lastly, we can prove (P3.3) and (P3.4) by (F1) and (F2) with $v_{31} = v_1$ (F3.5) and $v_{32} = v_2$ (F3.6). Having proven all premises (P3.1-4) from (IH3.1) we obtain:

$$\begin{aligned}
& (v_{31}, v_{32}) \in \mathcal{V}[\![\tau]\!]^{\rho_3} \\
& \equiv \langle \text{By (F3.5-6)} \rangle \\
& (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^{\rho_3} \\
& \equiv \langle \text{By Lemma B.8 knowing that } j \notin \Psi.1 \Rightarrow j \notin \mathbf{FV}(\tau) \rangle \\
& (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^{\rho_3 \setminus j} \\
& \equiv \langle \text{By def of } \rho_3 \text{ and } (_ \setminus _) \rangle \\
& (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^\rho
\end{aligned}$$

Which is precisely our goal (C').

- **Case** [T.NUM_R]: $\Psi; \mathcal{C}; \Gamma \vdash \mathbf{N} e' : \text{Rel } d \mathbb{R}$

$$\begin{aligned}
& \text{Goal} : \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash \mathbf{N} e', \gamma_2 \vdash \mathbf{N} e') \in \mathcal{E}[\![\text{Rel } d \mathbb{R}]\!]_\Gamma^\rho \\
& \equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \quad \gamma_1 \vdash \mathbf{N} e' \Downarrow v_1 \wedge \tag{H4} \\
& \quad \gamma_2 \vdash \mathbf{N} e' \Downarrow v_2 \tag{H5} \\
& \quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\![\text{Rel } d \mathbb{R}]\!]^\rho \tag{C'}
\end{aligned}$$

Consider the IH on $\Psi; \mathcal{C}; \Gamma \vdash e' : \mathbb{R}$

$$\begin{aligned}
& \forall \rho', \gamma'_1, \gamma'_2. \Psi \subseteq \text{dom}(\rho') \wedge \rho' \models \mathcal{C} \wedge (\gamma'_1, \gamma'_2) \in \mathcal{S}[\![\Gamma]\!]^{\rho'} \\
& \Rightarrow (\gamma'_1 \vdash e', \gamma'_2 \vdash e') \in \mathcal{E}[\![\mathbb{R}]\!]_\Gamma^{\rho'} \\
& \equiv \forall \rho', \gamma'_1, \gamma'_2, v'_1, v'_2. \\
& \quad \Psi \subseteq \text{dom}(\rho') \wedge \rho' \models \mathcal{C} \tag{P1} \\
& \quad \wedge (\gamma'_1, \gamma'_2) \in \mathcal{S}[\![\Gamma]\!]^{\rho'} \tag{P2} \\
& \quad \wedge \gamma'_1 \vdash e' \Downarrow v'_1 \tag{P3} \\
& \quad \wedge \gamma'_2 \vdash e' \Downarrow v'_2 \tag{P4} \\
& \quad \Rightarrow (v'_1, v'_2) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho'} \tag{IH}
\end{aligned}$$

Let $\rho' = \rho$, $\gamma'_1 = \gamma_1$, and $\gamma'_2 = \gamma_2$, then (P1) and (P2) are proven by (H2) and (H3), respectively.

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\gamma_1 \vdash \mathbf{N} e' \Downarrow v_1 \Rightarrow \gamma_1 \vdash e' \Downarrow v_1^* \wedge v_1 = \mathbf{N} v_1^* \tag{F1}$$

$$\gamma_2 \vdash \mathbf{N} e' \Downarrow v_2 \Rightarrow \gamma_2 \vdash e' \Downarrow v_2^* \wedge v_2 = \mathbf{N} v_2^* \tag{F2}$$

Let $v'_1 = v_1^*$ (F3) and $v'_2 = v_2^*$ (F4) so that (P3) and (P4) are proven by (F1-4). All preconditions for (IH) are fulfilled, then $(v'_1, v'_2) \in \mathcal{V}[\![\mathbb{R}]\!]^{\rho'}$; which, by definition of $\mathcal{V}[\![__]^\rho$ at \mathbb{R} , implies that $v'_1 \equiv v'_2$ (F5).

Now, recall our objective (C'):

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\llbracket \text{Rel } d \text{ Rel} \rrbracket^\rho \\
& \equiv \langle \text{By (F1-4)} \rangle \\
& \quad (\mathbf{N} v'_1, \mathbf{N} v'_2) \in \mathcal{V}[\llbracket \text{Rel } d \text{ Rel} \rrbracket^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[\llbracket _ \rrbracket^\rho \text{ at Rel } d \text{ Rel} \rangle \\
& \quad (\mathbf{N} v'_1, \mathbf{N} v'_2) \in \mathcal{D}[\llbracket \mathbb{R} \rrbracket_d^\rho \\
& \equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_d^\rho \text{ at Rel} \rangle \\
& \quad |v'_1 - v'_2| \leq \llbracket d \rrbracket_{\rho, d} \\
& \equiv \langle \text{By (F5)} \rangle \\
& \quad 0 \leq \llbracket d \rrbracket_{\rho, d} \tag{C''}
\end{aligned}$$

For any distance expression d and assignment ρ, d , the interpretation $\llbracket d \rrbracket_{\rho, d} \in \mathbb{R}_{\geq 0}$, then it must be the case that $0 \leq \llbracket d \rrbracket_{\rho, d}$.

- **Case [T.ADD]_R:** $\Psi, i; \mathcal{C} \wedge i = (d_1 + d_2); \Gamma \vdash e_1 + e_2 : \text{Rel } i \text{ Rel}$

$$\begin{aligned}
& \text{Goal : } \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash (e_1 + e_2), \gamma_2 \vdash (e_1 + e_2)) \in \mathcal{E}[\llbracket \text{Rel } d \text{ Rel} \rrbracket_\Gamma^\rho \\
& \equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \quad \gamma_1 \vdash (e_1 + e_2) \Downarrow v_1 \wedge \tag{H4} \\
& \quad \gamma_2 \vdash (e_1 + e_2) \Downarrow v_2 \tag{H5} \\
& \quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\llbracket \text{Rel } d \text{ Rel} \rrbracket^\rho \tag{C'}
\end{aligned}$$

Consider the IH on both sub-expressions

$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \text{ Rel}$

$$\begin{aligned}
& \forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\llbracket \Gamma \rrbracket^{\rho_1} \\
& \quad \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\llbracket \text{Rel } d_1 \text{ Rel} \rrbracket_\Gamma^{\rho_1} \\
& \equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\
& \quad \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \tag{P1.1} \\
& \quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\llbracket \Gamma \rrbracket^{\rho_1} \tag{P1.2} \\
& \quad \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \tag{P1.3} \\
& \quad \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \tag{P1.4} \\
& \quad \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\llbracket \text{Rel } d_1 \text{ Rel} \rrbracket^{\rho_1} \tag{IH1.1}
\end{aligned}$$

$$\Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \mathbb{R}$$

$$\begin{aligned} & \forall \rho_2, \gamma_{21}, \gamma_{22}. \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\Gamma]^{\rho_2} \\ & \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\text{Rel } d_2 \mathbb{R}]_{\Gamma}^{\rho_2} \\ \equiv & \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\ & \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \tag{P2.1} \\ & \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\Gamma]^{\rho_2} \tag{P2.2} \\ & \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \tag{P2.3} \\ & \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \tag{P2.4} \\ & \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\text{Rel } d_2 \mathbb{R}]^{\rho_2} \tag{IH2.1} \end{aligned}$$

Let $\rho_1 = \rho_2 = \rho$ (F1), then we can prove (P1.1) and (P2.1) by (H2) since:

$$\begin{aligned} & (\Psi, i) \subseteq \text{dom}(\rho) \Rightarrow \Psi \subseteq \text{dom}(\rho) \\ & \rho \models \mathcal{C}, i = (d_1 + d_2) \Rightarrow \rho \models \mathcal{C} \end{aligned}$$

Let $\gamma_{11} = \gamma_{21} = \gamma_1$ (F1.1) and $\gamma_{12} = \gamma_{22} = \gamma_2$ (F2.1). Then, by (H3), (F1), (F1.1), and (F2.1) we can prove (P1.2) and (P2.2), respectively.

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\begin{aligned} & \gamma_1 \vdash e_1 + e_2 \Downarrow v_1 \\ & \Rightarrow \gamma_1 \vdash e_1 \Downarrow \mathbf{N} v'_{11} \wedge \gamma_1 \vdash e_2 \Downarrow \mathbf{N} v'_{21} \wedge v_1 = \mathbf{N} (v'_{11} + v'_{21}) \tag{F1.2} \end{aligned}$$

$$\begin{aligned} & \gamma_2 \vdash e_1 + e_2 \Downarrow v_2 \\ & \Rightarrow \gamma_2 \vdash e_1 \Downarrow \mathbf{N} v'_{12} \wedge \gamma_2 \vdash e_2 \Downarrow \mathbf{N} v'_{22} \wedge v_2 = \mathbf{N} (v'_{12} + v'_{22}) \tag{F2.2} \end{aligned}$$

Let $v_{11} = \mathbf{N} v'_{11}$ (F1.3) and $v_{12} = \mathbf{N} v'_{12}$ (F1.4), then (P1.3) and (P1.4) are proven by (F1.1-4) and F(2.1-2). Similarly, making $v_{21} = \mathbf{N} v'_{21}$ (F1.3) and $v_{22} = \mathbf{N} v'_{22}$ (F2.4), we can prove (P2.3) and (P2.4) by (F1.1-2) and F(2.1-4).

Having fulfilled the preconditions for (IH1.1) and (IH2.1) we get:

$$\begin{aligned} & (v_{11}, v_{12}) \in \mathcal{V}[\text{Rel } d_1 \mathbb{R}]^{\rho} \wedge (v_{21}, v_{22}) \in \mathcal{V}[\text{Rel } d_2 \mathbb{R}]^{\rho} \\ \equiv & \langle \text{By (F1.3-4), (F2.3-4)} \rangle \\ & (\mathbf{N} v'_{11}, \mathbf{N} v'_{12}) \in \mathcal{V}[\text{Rel } d_1 \mathbb{R}]^{\rho} \wedge (\mathbf{N} v'_{21}, \mathbf{N} v'_{22}) \in \mathcal{V}[\text{Rel } d_2 \mathbb{R}]^{\rho} \\ \equiv & \langle \text{By def of } \mathcal{V}[_]^{\rho} \text{ at Rel } d \mathbb{R} \rangle \\ & (\mathbf{N} v'_{11}, \mathbf{N} v'_{12}) \in \mathcal{D}[\mathbb{R}]_{d_1}^{\rho} \wedge (\mathbf{N} v'_{21}, \mathbf{N} v'_{22}) \in \mathcal{D}[\mathbb{R}]_{d_2}^{\rho} \\ \equiv & \langle \text{By def of } \mathcal{D}[_]_d^{\rho} \text{ at } \mathbb{R} \rangle \\ & |v'_{11} - v'_{12}| \leq \llbracket d_1 \rrbracket_{\rho, d} \wedge |v'_{21} - v'_{22}| \leq \llbracket d_2 \rrbracket_{\rho, d} \tag{F2} \end{aligned}$$

Recall our objective (C'):

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\llbracket \text{Rel } i \text{ } \mathbb{R} \rrbracket^\rho \\
& \equiv \langle \text{By (F1.2), (F2.2)} \rangle \\
& \quad (\mathbf{N}(v'_{11} + v'_{21}), \mathbf{N}(v'_{12} + v'_{22})) \in \mathcal{V}[\llbracket \text{Rel } i \text{ } \mathbb{R} \rrbracket^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[\llbracket _ \rrbracket^\rho \text{ at Rel } d \text{ } \mathbb{R} \rangle \\
& \quad (\mathbf{N}(v'_{11} + v'_{21}), \mathbf{N}(v'_{12} + v'_{22})) \in \mathcal{D}[\llbracket \mathbb{R} \rrbracket_i^\rho \\
& \equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_i^\rho \text{ at } \mathbb{R} \rangle \\
& \quad |(v'_{11} + v'_{21}) - (v'_{12} + v'_{22})| \leq \llbracket i \rrbracket_{\rho, d} \\
& \equiv \langle \text{Since } \rho \models \mathcal{C}, i = (d_1 + d_2) \rangle \\
& \quad |(v'_{11} + v'_{21}) - (v'_{12} + v'_{22})| \leq \llbracket d_1 + d_2 \rrbracket_{\rho, d} \\
& \equiv \langle \text{By def of } \llbracket _ \rrbracket_{\rho, d} \rangle \\
& \quad |(v'_{11} + v'_{21}) - (v'_{12} + v'_{22})| \leq \llbracket d_1 \rrbracket_{\rho, d} + \llbracket d_2 \rrbracket_{\rho, d} \quad (\text{C''})
\end{aligned}$$

By triangle inequality and (F2)

$$|(v'_{11} - v'_{12}) + (v'_{21} - v'_{22})| \leq |v'_{11} - v'_{12}| + |v'_{21} - v'_{22}| \leq \llbracket d_1 \rrbracket_{\rho, d} + \llbracket d_2 \rrbracket_{\rho, d}$$

Since $|(v'_{11} - v'_{12}) + (v'_{21} - v'_{22})| \equiv |(v'_{11} + v'_{21}) - (v'_{12} + v'_{22})|$ then

$$\begin{aligned}
& |(v'_{11} + v'_{21}) - (v'_{12} + v'_{22})| \leq |v'_{11} - v'_{12}| + |v'_{21} - v'_{22}| \leq \llbracket d_1 \rrbracket_{\rho, d} + \llbracket d_2 \rrbracket_{\rho, d} \\
& \equiv \langle \text{By transitivity} \rangle \\
& |(v'_{11} + v'_{21}) - (v'_{12} + v'_{22})| \leq \llbracket d_1 \rrbracket_{\rho, d} + \llbracket d_2 \rrbracket_{\rho, d}
\end{aligned}$$

Proving (C'') and consequently (C') and (C).

- **Case** $[\text{T.PAIR}_R]$: $\Psi, i; \mathcal{C} \wedge i = (d_1 + d_2); \Gamma \vdash (e_1, e_2) : \text{Rel } i (\sigma_1 \times \sigma_2)$

$$\begin{aligned}
& \text{Goal : } \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash (e_1, e_2), \gamma_2 \vdash (e_1, e_2)) \in \mathcal{E}[\llbracket \text{Rel } i (\sigma_1 \times \sigma_2) \rrbracket_\Gamma^\rho \\
& \equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \quad \gamma_1 \vdash (e_1, e_2) \Downarrow v_1 \wedge \quad (\text{H4}) \\
& \quad \gamma_2 \vdash (e_1, e_2) \Downarrow v_2 \quad (\text{H5}) \\
& \quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\llbracket \text{Rel } i (\sigma_1 \times \sigma_2) \rrbracket^\rho \quad (\text{C'})
\end{aligned}$$

Consider the IH on both sub-expressions

$$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \sigma_1$$

$$\begin{aligned} & \forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\Gamma]^{\rho_1} \\ & \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\text{Rel } d_1 \sigma_1]_{\Gamma}^{\rho_1} \\ & \equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\ & \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \tag{P1.1} \\ & \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\Gamma]^{\rho_1} \tag{P1.2} \\ & \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \tag{P1.3} \\ & \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \tag{P1.4} \\ & \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\text{Rel } d_1 \sigma_1]^{\rho_1} \tag{IH1.1} \end{aligned}$$

$$\Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \sigma_2$$

$$\begin{aligned} & \forall \rho_2, \gamma_{21}, \gamma_{22}. \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\Gamma]^{\rho_2} \\ & \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\text{Rel } d_2 \sigma_2]_{\Gamma}^{\rho_2} \\ & \equiv \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\ & \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \tag{P2.1} \\ & \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\Gamma]^{\rho_2} \tag{P2.2} \\ & \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \tag{P2.3} \\ & \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \tag{P2.4} \\ & \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\text{Rel } d_2 \sigma_2]^{\rho_2} \tag{IH2.1} \end{aligned}$$

Let $\rho_1 = \rho_2 = \rho$ (F1), then we can prove (P1.1) and (P2.1) by (H2):

$$(\Psi, i) \subseteq \text{dom}(\rho) \Rightarrow \Psi \subseteq \text{dom}(\rho) \tag{F2}$$

$$\rho \models \mathcal{C}, i = (d_1 + d_2) \Rightarrow \rho \models \mathcal{C} \tag{F3}$$

Let $\gamma_{11} = \gamma_{21} = \gamma_1$ (F1.1) and $\gamma_{12} = \gamma_{22} = \gamma_2$ (F2.1). Then, by (H3), (F1), (F1.1), and (F2.1) we can prove (P1.2) and (P2.2), respectively.

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\begin{aligned} & \gamma_1 \vdash (e_1, e_2) \Downarrow v_1 \\ & \Rightarrow v_1 = (v'_{11}, v'_{21}) \wedge \gamma_1 \vdash e_1 \Downarrow v'_{11} \wedge \gamma_1 \vdash e_2 \Downarrow v'_{21} \tag{F1.2} \end{aligned}$$

$$\begin{aligned} & \gamma_2 \vdash (e_1, e_2) \Downarrow v_2 \\ & \Rightarrow v_2 = (v'_{12}, v'_{22}) \wedge \gamma_2 \vdash e_1 \Downarrow v'_{12} \wedge \gamma_2 \vdash e_2 \Downarrow v'_{22} \tag{F2.2} \end{aligned}$$

Let $v_{11} = v'_{11}$ (F1.3) and $v_{12} = v'_{12}$ (F1.4), then (P1.3) and (P1.4) are proven by (F1.1-4) and F(2.1-2). Similarly, making $v_{21} = v'_{21}$ (F1.3) and $v_{22} = v'_{22}$ (F2.4), we can prove (P2.3) and (P2.4) by (F1.1-2) and F(2.1-4).

Having fulfilled the preconditions for (IH1.1) and (IH2.1) we have:

$$\begin{aligned}
& (v_{11}, v_{12}) \in \mathcal{V}[\![\text{Rel } d_1 \sigma_1]\!]^\rho \wedge (v_{21}, v_{22}) \in \mathcal{V}[\![\text{Rel } d_2 \sigma_2]\!]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[\![_\cdot]\!]^\rho \text{ at Rel } d \sigma \rangle \\
& (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d_2}^\rho
\end{aligned} \tag{F4}$$

Recall our objective (C')

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\![\text{Rel } i (\sigma_1 \times \sigma_2)]\!]^\rho \\
& \equiv \langle \text{By F(1.2-4) and F(2.2-4)} \rangle \\
& ((v_{11}, v_{21}), (v_{12}, v_{22})) \in \mathcal{V}[\![\text{Rel } i (\sigma_1 \times \sigma_2)]\!]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[\![_\cdot]\!]^\rho \text{ at Rel } i (\sigma_1 \times \sigma_2) \rangle \\
& ((v_{11}, v_{21}), (v_{12}, v_{22})) \in \mathcal{D}[\![\sigma_1 \times \sigma_2]\!]_i^\rho \\
& \exists d'_1, d'_2. \rho.d \models i = d'_1 + d'_2 \\
& \wedge (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma_1]\!]_{d'_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\sigma_2]\!]_{d'_2}^\rho
\end{aligned} \tag{C''}$$

By (H2) we know that $\rho \models \mathcal{C}, i = (d_1 + d_2)$ then $\rho \models i = (d_1 + d_2)$, in particular, $\rho.d \models i = (d_1 + d_2)$ (F5). Choosing $d'_1 = d_1$ and $d'_2 = d_2$ we can prove (C'') by (F4) and (F5).

- **Case [T.LET_R]:** $\Psi; \mathcal{C}; \Gamma \vdash \text{let } (x, y) = e_1 \text{ in } e_2 : \tau$

Goal : $\forall \rho, \gamma_1, \gamma_2.$

$$\begin{aligned}
& (\gamma_1 \vdash \text{let } (x, y) = e_1 \text{ in } e_2, \gamma_2 \vdash \text{let } (x, y) = e_1 \text{ in } e_2) \in \mathcal{E}[\![\tau]\!]_\Gamma^\rho \\
& \equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \gamma_1 \vdash \text{let } (x, y) = e_1 \text{ in } e_2 \Downarrow v_1 \wedge \\
& \gamma_2 \vdash \text{let } (x, y) = e_1 \text{ in } e_2 \Downarrow v_2 \\
& \Rightarrow (v_1, v_2) \in \mathcal{V}[\![\tau]\!]^\rho
\end{aligned}
\tag{H4}$$

(H5)

(C')

Consider the IH on e_1

$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d (\sigma_1 \times \sigma_2)$

$$\begin{aligned}
& \forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\
& \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\![\text{Rel } d (\sigma_1 \times \sigma_2)]\!]_{\Gamma}^{\rho_1} \\
& \equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\
& \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \\
& \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\
& \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \\
& \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \\
& \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\text{Rel } d (\sigma_1 \times \sigma_2)]\!]^{\rho_1}
\end{aligned}
\tag{P1.1}$$

(P1.2)

(P1.3)

(P1.4)

(IH1.1)

Let $\rho_1 = \rho$ (F1.1), $\gamma_{11} = \gamma_1 \wedge \gamma_{12} = \gamma_2$ (F1.2); then (P1.1) is proven by (H2) while (P1.2) is proven by (F1.1-2) together with (H3).

By (H4), (H5) and cases on the evaluation relation ($_ \Downarrow _$) we have:

$$\begin{aligned} \gamma_1 \vdash \text{let } (x, y) = e_1 \text{ in } e_2 \Downarrow v_1 \\ \Rightarrow \gamma_1 \vdash e_1 \Downarrow (v'_{11}, v'_{21}) \wedge \gamma_1[x := v'_{11}][y := v'_{21}] \vdash e_2 \Downarrow v_1 \end{aligned} \quad (\text{F1.2})$$

$$\begin{aligned} \gamma_2 \vdash \text{let } (x, y) = e_1 \text{ in } e_2 \Downarrow v_2 \\ \Rightarrow \gamma_2 \vdash e_1 \Downarrow (v'_{12}, v'_{22}) \wedge \gamma_2[x := v'_{12}][y := v'_{22}] \vdash e_2 \Downarrow v_2 \end{aligned} \quad (\text{F2.2})$$

If we take $v_{11} = (v'_{11}, v'_{21}) \wedge v_{12} = (v'_{12}, v'_{22})$ (F1.3) we can prove (P1.3) and (P1.4) by (F1.2-3) and (F2.2). Having proven all the premises (P1.1-4) we obtain

$$\begin{aligned} (v_{11}, v_{12}) &\in \mathcal{V}[\llbracket \text{Rel } d \ (\sigma_1 \times \sigma_2) \rrbracket]^\rho \\ &\equiv \langle \text{By F(1.3)} \rangle \\ &\quad ((v'_{11}, v'_{21}), (v'_{12}, v'_{22})) \in \mathcal{V}[\llbracket \text{Rel } d \ (\sigma_1 \times \sigma_2) \rrbracket]^\rho \\ &\equiv \langle \text{By def of } \mathcal{V}[\llbracket _ \rrbracket]^\rho \text{ at Rel } d \ \sigma \rangle \\ &\quad ((v'_{11}, v'_{21}), (v'_{12}, v'_{22})) \in \mathcal{D}[\llbracket \sigma_1 \times \sigma_2 \rrbracket_d]^\rho \\ &\equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_d]^\rho \text{ at } (\sigma_1 \times \sigma_2) \rangle \\ &\quad \exists d'_1, d'_2. \rho.d \models d = (d'_1 + d'_2) \\ &\quad \wedge (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma_1 \rrbracket_{d'_1}^\rho] \wedge (v'_{21}, v'_{22}) \in \mathcal{D}[\llbracket \sigma_2 \rrbracket_{d'_2}^\rho] \\ &\equiv \langle \text{By Lemma B.5 with } \llbracket d'_1 \rrbracket_{\rho.d} = r_1 \text{ and } \llbracket d'_2 \rrbracket_{\rho.d} = r_2 \rangle \\ &\quad \exists r_1, r_2. \emptyset \models d = (r_1 + r_2) \\ &\quad \wedge (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma_1 \rrbracket_{r_1}^{(\emptyset, \rho, 1)}] \wedge (v'_{21}, v'_{22}) \in \mathcal{D}[\llbracket \sigma_2 \rrbracket_{r_2}^{(\emptyset, \rho, 1)}] \end{aligned} \quad (\text{IH1.2})$$

Now lets consider the IH on e_2 :

$\Psi, i_1, i_2; \mathcal{C} \wedge d = (i_1 + i_2); \Gamma, x : \text{Rel } i_1 \ \sigma_1, y : \text{Rel } i_2 \ \sigma_2 \vdash e_2 : \tau$ with $i_1, i_2 \notin \Psi.d$

$$\begin{aligned} &\forall \rho_2, \gamma_{21}, \gamma_{22}. \\ &\quad (\Psi, i_1, i_2) \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C}, d = (i_1 + i_2) \\ &\quad \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\llbracket \Gamma, x : \text{Rel } i_1 \ \sigma_1, y : \text{Rel } i_2 \ \sigma_2 \rrbracket]^\rho \\ &\quad \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\llbracket \tau \rrbracket_{(\Gamma, x:\text{Rel } i_1 \ \sigma_1, y:\text{Rel } i_2 \ \sigma_2)}^{\rho_2}] \\ &\equiv \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\ &\quad (\Psi, i_1, i_2) \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C}, d = (i_1 + i_2) \quad (\text{P2.1}) \\ &\quad \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\llbracket \Gamma, x : \text{Rel } i_1 \ \sigma_1, y : \text{Rel } i_2 \ \sigma_2 \rrbracket]^\rho \quad (\text{P2.2}) \\ &\quad \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \quad (\text{P2.3}) \\ &\quad \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \quad (\text{P2.4}) \\ &\quad \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\llbracket \tau \rrbracket]^\rho \quad (\text{IH2.1}) \end{aligned}$$

$$\text{Let } \rho_2 = \rho[i_1 := r_1][i_2 := r_2] \quad \text{with } r_1, r_2 \text{ from (IH1.2)} \quad (\text{F2.1})$$

$$\gamma_{21} = \gamma_1[x := v'_{11}][y := v'_{21}] \quad (\text{F2.2})$$

$$\gamma_{22} = \gamma_2[x := v'_{12}][y := v'_{22}] \quad (\text{F2.3})$$

We can prove (P2.1) by (H2), definition of $(_ \subseteq _)$, $(_ \models _)$ and (F2.1). Since $(\gamma_1, \gamma_2) \in \mathcal{S}[\Gamma]^\rho$, to prove $(\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\Gamma, x : \text{Rel } i_1 \sigma_1, y : \text{Rel } i_2 \sigma_2]^{\rho_2}$ (P2.2) we need to see that

$$\begin{aligned} & (\gamma_{21}(x), \gamma_{22}(x)) \in \mathcal{V}[\text{Rel } i_1 \sigma_1]^{\rho_2} \wedge (\gamma_{21}(y), \gamma_{22}(y)) \in \mathcal{V}[\text{Rel } i_2 \sigma_2]^{\rho_2} \\ & \equiv \langle \text{By (F2.2-3)} \rangle \\ & (v'_{11}, v'_{12}) \in \mathcal{V}[\text{Rel } i_1 \sigma_1]^{\rho_2} \wedge (v'_{21}, v'_{22}) \in \mathcal{V}[\text{Rel } i_2 \sigma_2]^{\rho_2} \\ & \equiv \langle \text{By def of } \mathcal{V}[_]^\rho \text{ at Rel } d \sigma \rangle \\ & (v'_{11}, v'_{12}) \in \mathcal{D}[\sigma_1]_{i_1}^{\rho_2} \wedge (v'_{21}, v'_{22}) \in \mathcal{D}[\sigma_2]_{i_2}^{\rho_2} \\ & \equiv \langle \text{By Lemma B.5 knowing that } \llbracket i_1 \rrbracket_{\rho, d} = r_1 \text{ and } \llbracket i_2 \rrbracket_{\rho, d} = r_2 \text{ (F2.1)} \rangle \\ & (v'_{11}, v'_{12}) \in \mathcal{D}[\sigma_1]_{r_1}^{(\emptyset, \rho, 1)} \wedge (v'_{21}, v'_{22}) \in \mathcal{D}[\sigma_2]_{r_2}^{(\emptyset, \rho, 1)} \end{aligned}$$

which holds by (IH1.2).

Lastly, we can prove (P2.3) and (P2.4) by (F1.2) and (F2.2) with $v_{21} = v_1$ (F2.4) and $v_{22} = v_2$ (F2.5). Having proven all premises (P2.1-4) from (IH2.1) we obtain

$$(v_{21}, v_{22}) \in \mathcal{V}[\tau]^{\rho_2} \quad (\text{IH2.2})$$

Since $i_1, i_2 \notin \Psi.d$ it must be the case that $i_1, i_2 \notin \mathbf{FV}(\tau)$. By Lemma B.8 we can safely remove the substitutions for i_1 and i_2 from ρ_2 , this is:

$$\begin{aligned} & (v_{21}, v_{22}) \in \mathcal{V}[\tau]^{\rho_2} \\ & \equiv \langle \text{By Lemma B.8} \rangle \\ & (v_{21}, v_{22}) \in \mathcal{V}[\tau]^{\rho_2 \setminus \{i_1, i_2\}} \\ & \equiv \langle \text{By def of } \rho_2 \text{ and } (_ \setminus _) \rangle \\ & (v_{21}, v_{22}) \in \mathcal{V}[\tau]^\rho \\ & \equiv \langle \text{By (F2.4-5)} \rangle \\ & (v_1, v_2) \in \mathcal{V}[\tau]^\rho \end{aligned}$$

Which is precisely our (C') goal.

- **Case [T.NIL_R]:** $\Psi, j; \mathcal{C} \wedge j = 0; \Gamma \vdash [] : \text{Rel } d \vec{\sigma}_j$

$$\begin{aligned}
\text{Goal} &: \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash [], \gamma_2 \vdash []) \in \mathcal{E}[\llbracket \text{Rel } d \vec{\sigma}_j \rrbracket_\Gamma]^\rho \\
&\equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
&\quad \gamma_1 \vdash [] \Downarrow v_1 \wedge \quad (H4) \\
&\quad \gamma_2 \vdash [] \Downarrow v_2 \quad (H5) \\
&\Rightarrow (v_1, v_2) \in \mathcal{V}[\llbracket \text{Rel } d \vec{\sigma}_j \rrbracket]^\rho \quad (C')
\end{aligned}$$

By **E.NIL_R** and cases on the evaluation relation $(_ \Downarrow _)$ we know $\gamma_1 \vdash [] \Downarrow []$, and $\gamma_2 \vdash [] \Downarrow []$. Let $v_1 = v_2 = []$, then we need to show that $([], []) \in \mathcal{V}[\llbracket \text{Rel } d \vec{\sigma}_j \rrbracket]^\rho$, which, by definition of $\mathcal{V}[\llbracket _ \rrbracket]^\rho$ at $\text{Rel } d \sigma$ is equivalent to show that $([], []) \in \mathcal{D}[\llbracket \vec{\sigma}_j \rrbracket_d]^\rho$. By (H2) we know that $\rho \models \mathcal{C} \wedge j = 0$, then $\llbracket j \rrbracket_{\rho.1} = 0$ (by definition of $(_ \models _)$ and $\llbracket _ \rrbracket_{\rho.1}$). With this, we can prove that $([], []) \in \mathcal{D}[\llbracket \vec{\sigma}_0 \rrbracket_d]^\rho$ by definition of $\mathcal{D}[\llbracket _ \rrbracket_d]^\rho$ at $\vec{\sigma}_0$.

- **Case [T.CONSR_R]:**

$$\Psi, i, j; \mathcal{C} \wedge j = (l + 1) \wedge i = (d_1 + d_2); \Gamma \vdash e_1 :: e_2 : \text{Rel } i \vec{\sigma}_j$$

$$\begin{aligned}
\text{Goal} &: \forall \rho, \gamma_1, \gamma_2. (\gamma_1 \vdash e_1 :: e_2, \gamma_2 \vdash e_1 :: e_2) \in \mathcal{E}[\llbracket \text{Rel } i \vec{\sigma}_j \rrbracket_\Gamma]^\rho \\
&\equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
&\quad \gamma_1 \vdash e_1 :: e_2 \Downarrow v_1 \wedge \quad (H4) \\
&\quad \gamma_2 \vdash e_1 :: e_2 \Downarrow v_2 \quad (H5) \\
&\Rightarrow (v_1, v_2) \in \mathcal{V}[\llbracket \text{Rel } i \vec{\sigma}_j \rrbracket]^\rho \quad (C')
\end{aligned}$$

Consider the IH on both sub-expressions

$$\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d_1 \sigma$$

$$\begin{aligned}
&\forall \rho_1, \gamma_{11}, \gamma_{12}. \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\llbracket \Gamma \rrbracket]^{\rho_1} \\
&\Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\llbracket \text{Rel } d_1 \sigma \rrbracket_\Gamma]^{\rho_1} \\
&\equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\
&\quad \Psi \subseteq \text{dom}(\rho_1) \wedge \rho_1 \models \mathcal{C} \quad (P1.1) \\
&\quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\llbracket \Gamma \rrbracket]^{\rho_1} \quad (P1.2) \\
&\quad \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \quad (P1.3) \\
&\quad \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \quad (P1.4) \\
&\Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\llbracket \text{Rel } d_1 \sigma \rrbracket]^{\rho_1} \quad (IH1.1)
\end{aligned}$$

$$\Psi; \mathcal{C}; \Gamma \vdash e_2 : \text{Rel } d_2 \vec{\sigma}_l$$

$$\begin{aligned} & \forall \rho_2, \gamma_{21}, \gamma_{22}. \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \\ & \Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\![\text{Rel } d_2 \vec{\sigma}_l]\!]_{\Gamma}^{\rho_2} \\ & \equiv \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}. \\ & \Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \tag{P2.1} \\ & \wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \tag{P2.2} \\ & \wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \tag{P2.3} \\ & \wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \tag{P2.4} \\ & \Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\text{Rel } d_2 \vec{\sigma}_l]\!]^{\rho_2} \tag{IH2.1} \end{aligned}$$

Let $\rho_1 = \rho_2 = \rho$ (F1), then we can prove (P1.1) and (P2.1) by (H2):

$$(\Psi, i) \subseteq \text{dom}(\rho) \Rightarrow \Psi \subseteq \text{dom}(\rho) \tag{F2}$$

$$\rho \models \mathcal{C}, i = (d_1 + d_2) \Rightarrow \rho \models \mathcal{C} \tag{F3}$$

Let $\gamma_{11} = \gamma_{21} = \gamma_1$ (F1.1) and $\gamma_{12} = \gamma_{22} = \gamma_2$ (F2.1). Then, by (H3), (F1), (F1.1), and (F2.1) we can prove (P1.2) and (P2.2), respectively.

By (H4), (H5), and cases on the evaluation relation ($_ \Downarrow _$):

$$\begin{aligned} & \gamma_1 \vdash e_1 :: e_2 \Downarrow v_1 \\ & \Rightarrow v_1 = v'_{11} :: v'_{21} \wedge \gamma_1 \vdash e_1 \Downarrow v'_{11} \wedge \gamma_1 \vdash e_2 \Downarrow v'_{21} \tag{F1.2} \end{aligned}$$

$$\begin{aligned} & \gamma_2 \vdash e_1 :: e_2 \Downarrow v_2 \\ & \Rightarrow v_2 = v'_{12} :: v'_{22} \wedge \gamma_2 \vdash e_1 \Downarrow v'_{12} \wedge \gamma_2 \vdash e_2 \Downarrow v'_{22} \tag{F2.2} \end{aligned}$$

Let $v_{11} = v'_{11}$ (F1.3) and $v_{12} = v'_{12}$ (F1.4), then (P1.3) and (P1.4) are proven by (F1.1-4) and F(2.1-2). Similarly, making $v_{21} = v'_{21}$ (F1.3) and $v_{22} = v'_{22}$ (F2.4), we can prove (P2.3) and (P2.4) by (F1.1-2) and F(2.1-4).

Having fulfilled the preconditions for (IH1.1) and (IH2.1) we have:

$$\begin{aligned} & (v_{11}, v_{12}) \in \mathcal{V}[\![\text{Rel } d_1 \sigma]\!]^\rho \wedge (v_{21}, v_{22}) \in \mathcal{V}[\![\text{Rel } d_2 \vec{\sigma}_l]\!]^\rho \\ & \equiv \langle \text{By def of } \mathcal{V}[\![_]\!]^\rho \text{ at Rel } d \sigma \rangle \\ & (v_{11}, v_{12}) \in \mathcal{D}[\![\sigma]\!]_{d_1}^\rho \wedge (v_{21}, v_{22}) \in \mathcal{D}[\![\vec{\sigma}_l]\!]_{d_2}^\rho \tag{F4} \end{aligned}$$

By (H2) we know that $\rho \models \mathcal{C} \wedge j = (l+1) \wedge i = (d_1 + d_2)$, then, by definition of ($_ \models _$) and $\llbracket j \rrbracket_{\rho,1}$, we know that $\llbracket j \rrbracket_{\rho,1} \equiv \llbracket l \rrbracket_{\rho,1} + 1$. Let $\llbracket l \rrbracket_{\rho,1} = m$ (with $m \in \mathbb{N}$), then $\llbracket j \rrbracket_{\rho,1} \equiv m + 1$ (F5).

Now, recall our objective (C')

$$\begin{aligned}
& (v_1, v_2) \in \mathcal{V}[\llbracket \text{Rel } i \ \vec{\sigma}_j \rrbracket]^\rho \\
& \equiv \langle \text{By F(1.2-4) and F(2.2-4)} \rangle \\
& \quad (v_{11} :: v_{21}, v_{12} :: v_{22}) \in \mathcal{V}[\llbracket \text{Rel } i \ \vec{\sigma}_j \rrbracket]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[\llbracket _ \rrbracket]^\rho \text{ at Rel } d \ \sigma \rangle \\
& \quad (v_{11} :: v_{21}, v_{12} :: v_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_j \rrbracket_i]^\rho \\
& \equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_i]^\rho \text{ at } \vec{\sigma}_i \text{ with } \llbracket l \rrbracket_{\rho,1} \neq 0 \rangle \\
& \quad \exists d'_1, d'_2. \rho.d \models i = d'_1 + d'_2 \\
& \quad \quad \wedge (v_{11}, v_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{d'_1}^\rho] \wedge (v_{21}, v_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{(\llbracket j \rrbracket_{\rho,1}-1)} \rrbracket_{d'_2}^\rho] \\
& \equiv \langle \text{By (F5) and subtraction} \rangle \\
& \quad \exists d'_1, d'_2. \rho.d \models i = d'_1 + d'_2 \\
& \quad \quad \wedge (v_{11}, v_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{d'_1}^\rho] \wedge (v_{21}, v_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_m \rrbracket_{d'_2}^\rho] \tag{C''}
\end{aligned}$$

By (H2) we know that $\rho \models \mathcal{C} \wedge j = (l+1) \wedge i = (d_1 + d_2)$ then $\rho.d \models i = (d_1 + d_2)$ (F5). Choosing $d'_1 = d_1$ and $d'_2 = d_2$ we can prove $(v_{11}, v_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{d'_1}^\rho]$ directly by (F4).

To prove $(v_{21}, v_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_m \rrbracket_{d'_2}^\rho]$ we use (F4)'s second conjunct, the assignment $\llbracket l \rrbracket_{\rho,1} = m$, and the fact that for any two values $(u_1, u_2) \in \mathcal{D}[\llbracket \vec{\sigma}_l \rrbracket_d^\rho]$ it must be the case that $(u_1, u_2) \in \mathcal{D}[\llbracket \vec{\sigma}_l \rrbracket_{\rho,1}^\rho]$.

- **Case** [T.CASE_R]: $\Psi; \mathcal{C}; \Gamma \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} : \tau$

Goal : $\forall \rho, \gamma_1, \gamma_2.$

$$\begin{aligned}
& (\gamma_1 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \\
& \quad , \gamma_1 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\}) \in \mathcal{E}[\llbracket \tau \rrbracket_\Gamma]^\rho \\
& \equiv \forall \rho, \gamma_1, \gamma_2, v_1, v_2. \\
& \quad \gamma_1 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \Downarrow v_1 \wedge \tag{H4} \\
& \quad \gamma_2 \vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \Downarrow v_2 \tag{H5} \\
& \quad \Rightarrow (v_1, v_2) \in \mathcal{V}[\llbracket \tau \rrbracket]^\rho \tag{C'}
\end{aligned}$$

By cases on the typing derivation of $\Psi; \mathcal{C}; \Gamma \vdash e_1 : \text{Rel } d \ \vec{\sigma}_l$ we obtain:

- **Case** [T.NIL_R]:

$$e_1 : \quad \Psi', j'; \mathcal{C}' \wedge j' = 0; \Gamma \vdash [_] : \text{Rel } d \ \vec{\sigma}_{j'}$$

In this case we have

$$\Psi \equiv \Psi', j' \quad (\text{D1})$$

$$\mathcal{C} \equiv \mathcal{C}' \wedge j' = 0 \quad (\text{D2})$$

$$l \equiv j' \quad (\text{D3})$$

$$e_1 \equiv [] \quad (\text{D4})$$

By (H4), (H5), (D4), and cases on the evaluation relation ($_ \Downarrow _$) we have:

$$\begin{aligned} \gamma_1 \vdash \text{case } e_1 \text{ of } \{[] \cdot e_2\} \{(x :: xs) \cdot e_3\} \Downarrow v_1 \\ \Rightarrow \gamma_1 \vdash e_1 \Downarrow [] \wedge \gamma_1 \vdash e_2 \Downarrow v_1 \end{aligned} \quad (\text{F1})$$

$$\begin{aligned} \gamma_2 \vdash \text{case } e_1 \text{ of } \{[] \cdot e_2\} \{(x :: xs) \cdot e_3\} \Downarrow v_2 \\ \Rightarrow \gamma_2 \vdash e_1 \Downarrow [] \wedge \gamma_2 \vdash e_2 \Downarrow v_2 \end{aligned} \quad (\text{F2})$$

Now, let's consider the IH on e_2 :

$\Psi; \mathcal{C} \wedge l = 0; \Gamma \vdash e_2 : \tau$

$\forall \rho_2, \gamma_{21}, \gamma_{22}.$

$$\Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge l = 0$$

$$\wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2}$$

$$\Rightarrow (\gamma_{21} \vdash e_2, \gamma_{22} \vdash e_2) \in \mathcal{E}[\![\tau]\!]_{\Gamma}^{\rho_2}$$

$\equiv \forall \rho_2, \gamma_{21}, \gamma_{22}, v_{21}, v_{22}.$

$$\Psi \subseteq \text{dom}(\rho_2) \wedge \rho_2 \models \mathcal{C} \wedge l = 0 \quad (\text{P2.1})$$

$$\wedge (\gamma_{21}, \gamma_{22}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_2} \quad (\text{P2.2})$$

$$\wedge \gamma_{21} \vdash e_2 \Downarrow v_{21} \quad (\text{P2.3})$$

$$\wedge \gamma_{22} \vdash e_2 \Downarrow v_{22} \quad (\text{P2.4})$$

$$\Rightarrow (v_{21}, v_{22}) \in \mathcal{V}[\![\tau]\!]^{\rho_2} \quad (\text{IH2.1})$$

Let $\rho_2 = \rho$ (F3), $\gamma_{21} = \gamma_1 \wedge \gamma_{22} = \gamma_2$ (F4); then (P2.1) is proven by (H2) given that $l \equiv j'$ (D3). Similarly, premise (P2.2) is proven by (F1-4) together with (H3).

Let $v_{21} = v_1$ and $v_{22} = v_2$, then we can prove (P2.3) and (P2.4) by (F1) and (F2), respectively. Having proven all the premises (P2.1-4) we obtain $(v_1, v_2) \in \mathcal{V}[\![\tau]\!]^{\rho}$, that is no less than our objective (C').

– **Case [T.CONSR_R]:**

$e_1 : \Psi', j', i; \mathcal{C}' \wedge j' = (l' + 1) \wedge i = (d_1 + d_2); \Gamma \vdash e'_1 :: es'_1 : \text{Rel } i \vec{\sigma}_{j'}$

In this case we have

$$\Psi \equiv \Psi', j', i \quad (\text{D1})$$

$$\mathcal{C} \equiv \mathcal{C}' \wedge j' = (l' + 1) \wedge i = (d_1 + d_2) \quad (\text{D2})$$

$$l \equiv j' \quad (\text{D3})$$

$$e_1 \equiv e'_1 :: es'_1 \quad (\text{D4})$$

$$d \equiv i \quad (\text{D5})$$

By (H4), (H5), (D4), and cases on the evaluation relation ($_ \Downarrow _$) we have:

$$\begin{aligned} \gamma_1 &\vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \Downarrow v_1 \\ &\Rightarrow \gamma_1 \vdash e_1 \Downarrow v'_{11} :: vs'_{21} \\ &\quad \wedge \gamma_1[x := v'_{11}][xs := vs'_{21}] \vdash e_3 \Downarrow v_1 \end{aligned} \quad (\text{F1})$$

$$\begin{aligned} \gamma_2 &\vdash \text{case } e_1 \text{ of } \{[_].e_2\}\{(x :: xs).e_3\} \Downarrow v_2 \\ &\Rightarrow \gamma_2 \vdash e_1 \Downarrow v'_{12} :: vs'_{22} \\ &\quad \wedge \gamma_2[x := v'_{12}][xs := vs'_{22}] \vdash e_3 \Downarrow v_2 \end{aligned} \quad (\text{F2})$$

Lets consider the IH on e_1 :

$$\begin{aligned} &\forall \rho_1, \gamma_{11}, \gamma_{12}. (\Psi', j', i) \subseteq \text{dom}(\rho_1) \\ &\quad \wedge \rho_1 \models \mathcal{C}' \wedge j' = (l' + 1) \wedge i = (d_1 + d_2) \\ &\quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \\ &\quad \Rightarrow (\gamma_{11} \vdash e_1, \gamma_{12} \vdash e_1) \in \mathcal{E}[\![\text{Rel } i \vec{\sigma}_{j'}]\!]_{\Gamma}^{\rho_1} \\ &\equiv \forall \rho_1, \gamma_{11}, \gamma_{12}, v_{11}, v_{12}. \\ &\quad (\Psi', j', i) \subseteq \text{dom}(\rho_1) \quad (\text{P1.1.1}) \\ &\quad \wedge \rho_1 \models \mathcal{C}' \wedge j' = (l' + 1) \wedge i = (d_1 + d_2) \quad (\text{P1.1.2}) \\ &\quad \wedge (\gamma_{11}, \gamma_{12}) \in \mathcal{S}[\![\Gamma]\!]^{\rho_1} \quad (\text{P1.2}) \\ &\quad \wedge \gamma_{11} \vdash e_1 \Downarrow v_{11} \quad (\text{P1.3}) \\ &\quad \wedge \gamma_{12} \vdash e_1 \Downarrow v_{12} \quad (\text{P1.4}) \\ &\quad \Rightarrow (v_{11}, v_{12}) \in \mathcal{V}[\![\text{Rel } i \vec{\sigma}_{j'}]\!]^{\rho_1} \quad (\text{IH1.1}) \end{aligned}$$

Let $\rho_1 = \rho$ (F1.1), $\gamma_{11} = \gamma_1 \wedge \gamma_{12} = \gamma_2$ (F1.2); then (P1.1.1-2) are proven by (H2), (D1), and (D2). Similarly, premise (P1.2) is proven by (F1.1-2), (D1), (D2), together with (H3).

If we take $v_{11} = v'_{11} :: vs'_{21}$ (F1.3) and $v_{12} = v'_{12} :: vs'_{22}$ (F1.4) we can prove (P1.3) and (P1.4) by (F1) and (F2). Having proven all the premises (P1.1-4) we obtain:

$$\begin{aligned}
& (v'_{11} :: vs'_{21}, v'_{12} :: vs'_{22}) \in \mathcal{V}[\llbracket \text{Rel } i \ \vec{\sigma}_j \rrbracket]^\rho \\
& \equiv \langle \text{By def of } \mathcal{V}[\llbracket _ \rrbracket]^\rho \text{ at Rel } d \ \sigma \rangle \\
& \quad (v'_{11} :: vs'_{21}, v'_{12} :: vs'_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{j'} \rrbracket_i]^\rho \\
& \equiv \langle \text{By def of } \mathcal{D}[\llbracket _ \rrbracket_d]^\rho \text{ at } \vec{\sigma}_i \text{ with } \llbracket l \rrbracket_{\rho,1} \neq 0 \rangle \\
& \quad \exists d'_1, d'_2. \rho.d \models i = d'_1 + d'_2 \\
& \quad \quad \wedge (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{d'_1}^\rho] \wedge (vs'_{21}, vs'_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{\langle \llbracket j' \rrbracket_{\rho,1}-1} \rrbracket]_{d'_2}^\rho \\
& \equiv \langle \text{By } \rho \models \mathcal{C} \text{ considering (D2), (D3)} \rangle \\
& \quad \exists d'_1, d'_2. \rho.d \models i = d'_1 + d'_2 \\
& \quad \quad \wedge (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{d'_1}^\rho] \wedge (vs'_{21}, vs'_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{\langle \llbracket l' \rrbracket_{\rho,1}+1-1} \rrbracket]_{d'_2}^\rho \\
& \equiv \langle \text{By simplification} \rangle \\
& \quad \exists d'_1, d'_2. \rho.d \models i = d'_1 + d'_2 \\
& \quad \quad \wedge (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{d'_1}^\rho] \wedge (vs'_{21}, vs'_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{\llbracket l' \rrbracket_{\rho,1}} \rrbracket]_{d'_2}^\rho \\
& \equiv \langle \text{By Lemma B.5 with } \llbracket d'_1 \rrbracket_{\rho,d} = r_2 \text{ and } \llbracket d'_2 \rrbracket_{\rho,d} = r_2 \rangle \\
& \quad \exists r_1, r_2. \emptyset \models \llbracket i \rrbracket_{\rho,d} = r_1 + r_2 \\
& \quad \quad \wedge (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{r_1}^{(\emptyset, \rho, 1)}] \\
& \quad \quad \wedge (vs'_{21}, vs'_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{\llbracket l' \rrbracket_{\rho,1}} \rrbracket_{r_2}^{(\emptyset, \rho, 1)}] \tag{IH1.2}
\end{aligned}$$

Now, lets consider the IH on e_3 :

$\Psi, i_1, i_2, j; \mathcal{C} \wedge l = (j+1) \wedge d = (i_1 + i_2); \Gamma, x : \text{Rel } i_1 \ \sigma, xs : \text{Rel } i_2 \ \vec{\sigma}_j \vdash e_3 : \tau$ with $i_1, i_2 \notin \Psi.d$ and $j \notin \Psi.l$

$$\begin{aligned}
& \forall \rho_3, \gamma_{31}, \gamma_{32}. \\
& \quad (\Psi, i_1, i_2, j) \subseteq \text{dom}(\rho_3) \wedge \rho_3 \models \mathcal{C} \wedge l = (j+1) \wedge d = (i_1 + i_2) \\
& \quad \wedge (\gamma_{31}, \gamma_{32}) \in \mathcal{S}[\llbracket \Gamma, x : \text{Rel } i_1 \ \sigma, xs : \text{Rel } i_2 \ \vec{\sigma}_j \rrbracket]_{\rho_3}^{\rho_3} \\
& \quad \Rightarrow (\gamma_{31} \vdash e_3, \gamma_{32} \vdash e_3) \in \mathcal{E}[\llbracket \tau \rrbracket_{(\Gamma, x : \text{Rel } i_1 \ \sigma, xs : \text{Rel } i_2 \ \vec{\sigma}_j)}^{\rho_3}] \\
& \equiv \forall \rho_3, \gamma_{31}, \gamma_{32}, v_{31}, v_{32}. \\
& \quad (\Psi, i_1, i_2, j) \subseteq \text{dom}(\rho_3) \tag{P3.1} \\
& \quad \wedge \rho_3 \models \mathcal{C} \wedge l = (j+1) \wedge d = (i_1 + i_2) \tag{P3.2} \\
& \quad \wedge (\gamma_{31}, \gamma_{32}) \in \mathcal{S}[\llbracket \Gamma, x : \text{Rel } i_1 \ \sigma, xs : \text{Rel } i_2 \ \vec{\sigma}_j \rrbracket]_{\rho_3}^{\rho_3} \tag{P3.3} \\
& \quad \wedge \gamma_{31} \vdash e_3 \Downarrow v_{31} \tag{P3.4} \\
& \quad \wedge \gamma_{32} \vdash e_3 \Downarrow v_{32} \tag{P3.5} \\
& \quad \Rightarrow (v_{31}, v_{32}) \in \mathcal{V}[\llbracket \tau \rrbracket]_{\rho_3}^{\rho_3} \tag{IH3.1}
\end{aligned}$$

$$\text{Let } \rho_3 = \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \quad (\text{F3.1})$$

$$\gamma_{31} = \gamma_1[x := v'_{11}][xs := vs'_{21}] \quad (\text{F3.2})$$

$$\gamma_{32} = \gamma_2[x := v'_{12}][xs := vs'_{22}] \quad (\text{F3.3})$$

We can prove (P3.1) by (H2) together with the definition of $(_ \subseteq _)$. Now, to see that (P3.2) holds we need to unfold the definition of $(_ \models _)$ as follows:

$$\begin{aligned} & \rho_3 \models C \wedge l = (j + 1) \wedge d = (i_1 + i_2) \\ \equiv & \langle \text{By (D2), (D3), and (D5)} \rangle \\ & \rho_3 \models C' \wedge j' = (l' + 1) \wedge l = (j + 1) \wedge i = (i_1 + i_2) \\ \equiv & \langle \text{By (F3.1)} \rangle \\ & \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models \\ & \quad C' \wedge j' = (l' + 1) \wedge l = (j + 1) \wedge i = (i_1 + i_2) \\ \equiv & \langle \text{By def of } (_ \models _) \rangle \\ & \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models C' \quad (\text{P3.2.1}) \\ & \wedge \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models j' = (l' + 1) \quad (\text{P3.2.2}) \\ & \wedge \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models j' = (j + 1) \quad (\text{P3.2.3}) \\ & \wedge \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models i = (i_1 + i_2) \quad (\text{P3.2.4}) \end{aligned}$$

Given that $\rho \models C' \wedge j' = (l' + 1)$ (by (H2) and (D2)) then it must be the case that $\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models C' \wedge j' = (l' + 1)$, proving (P3.2.1) and (P3.2.2). From the latter, we have that:

$$\begin{aligned} & \llbracket j' \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \\ & \quad \equiv \llbracket l' + 1 \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \\ \equiv & \langle \text{By def of } \llbracket _ \rrbracket_{\rho} \rangle \\ & \llbracket j' \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \\ & \quad \equiv \llbracket l' \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} + 1 \\ \equiv & \langle \text{Since } i_2, i_2, j \notin \text{vars}(l') \rangle \\ & \llbracket j' \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket l' \rrbracket_{\rho.1} + 1 \quad (\text{F3.4}) \end{aligned}$$

To prove (P3.2.3) consider the definition of $(_ \models _)$

$$\begin{aligned}
& \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models j' = (j + 1) \\
& \equiv \langle \text{By def of } (_ \models _) \rangle \\
& \quad \llbracket j' \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \\
& \quad \equiv \llbracket j + 1 \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \\
& \equiv \langle \text{By def of } \llbracket _ \rrbracket_{\rho} \rangle \\
& \quad \llbracket j' \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \\
& \quad \equiv \llbracket j \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} + 1 \\
& \equiv \langle \text{By def of } \llbracket _ \rrbracket_{\rho} \rangle \\
& \quad \llbracket j' \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket l' \rrbracket_{\rho.1} + 1
\end{aligned}$$

which is proven by (F3.4).

Lastly, we need satisfy (P3.2.4) this is

$$\begin{aligned}
& \rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}] \models i = (i_1 + i_2) \\
& \equiv \langle \text{By def of } (_ \models _) \rangle \\
& \quad \llbracket i \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \equiv \llbracket i_1 + i_2 \rrbracket_{\rho[i_1 := r_1][i_2 := r_2][j := \llbracket l' \rrbracket_{\rho.1}]} \\
& \equiv \langle \text{By removing free variables on both sides} \rangle \\
& \quad \llbracket i \rrbracket_{\rho} \equiv \llbracket i_1 + i_2 \rrbracket_{\rho[i_1 := r_1][i_2 := r_2]} \\
& \equiv \langle \text{By def of } \llbracket _ \rrbracket_{\rho} \rangle \\
& \quad \llbracket i \rrbracket_{\rho} \equiv r_1 + r_2 \\
& \equiv \langle \text{Since } i \text{ is a distance variable} \rangle \\
& \quad \llbracket i \rrbracket_{\rho.d} \equiv r_1 + r_2
\end{aligned}$$

which is proven by $\emptyset \models \llbracket i \rrbracket_{\rho.d} = r_1 + r_2$ from (IH1.2). With this we have completed all conditions to prove (P3.2).

To prove (P3.3) we need to see that:

$$\begin{aligned}
& (\gamma_{31}(x), \gamma_{32}(x)) \in \mathcal{V}[\llbracket \text{Rel } i_1 \ \sigma \rrbracket^{\rho_3} \wedge \\
& (\gamma_{31}(xs), \gamma_{32}(xs)) \in \mathcal{V}[\llbracket \text{Rel } i_2 \ \vec{\sigma}_j \rrbracket^{\rho_3} \\
& \equiv \langle \text{By (F3.1-3)} \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{V}[\llbracket \text{Rel } i_1 \ \sigma \rrbracket^{\rho[i_1:=r_1][i_2:=r_2][j:=\llbracket l' \rrbracket_{\rho,1}]} \wedge \\
& (vs'_{21}, vs'_{22}) \in \mathcal{V}[\llbracket \text{Rel } i_2 \ \vec{\sigma}_j \rrbracket^{\rho[i_1:=r_1][i_2:=r_2][j:=\llbracket l' \rrbracket_{\rho,1}]} \\
& \equiv \langle \text{By removing free variables (Lemma B.8) and expanding } \rho \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{V}[\llbracket \text{Rel } i_1 \ \sigma \rrbracket^{(\rho.d[i_1:=r_1], \rho, 1)} \wedge \\
& (vs'_{21}, vs'_{22}) \in \mathcal{V}[\llbracket \text{Rel } i_2 \ \vec{\sigma}_j \rrbracket^{(\rho.d[i_2:=r_2], \rho, 1)[j:=\llbracket l' \rrbracket_{\rho,1}]} \\
& \equiv \langle \text{By interpreting } j \text{ and knowing that } j \notin \mathbf{FV}(\sigma) \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{V}[\llbracket \text{Rel } i_1 \ \sigma \rrbracket^{(\rho.d[i_1:=r_1], \rho, 1)} \wedge \\
& (vs'_{21}, vs'_{22}) \in \mathcal{V}[\llbracket \text{Rel } i_2 \ \vec{\sigma}_{\llbracket l' \rrbracket_{\rho,1}} \rrbracket^{(\rho.d[i_2:=r_2], \rho, 1)} \\
& \equiv \langle \text{By def of } \mathcal{V}[\llbracket _ \rrbracket^\rho \text{ at Rel } d \ \sigma] \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{i_1}^{(\rho.d[i_1:=r_1], \rho, 1)} \wedge \\
& (vs'_{21}, vs'_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{\llbracket l' \rrbracket_{\rho,1}} \rrbracket_{i_2}^{(\rho.d[i_2:=r_2], \rho, 1)} \\
& \equiv \langle \text{By Lemma B.5} \rangle \\
& (v'_{11}, v'_{12}) \in \mathcal{D}[\llbracket \sigma \rrbracket_{r_1}^{(\emptyset, \rho, 1)} \wedge (vs'_{21}, vs'_{22}) \in \mathcal{D}[\llbracket \vec{\sigma}_{\llbracket l' \rrbracket_{\rho,1}} \rrbracket_{r_2}^{(\emptyset, \rho, 1)}
\end{aligned}$$

which holds directly by (IH1.2).

Lastly, we can prove (P3.4) and (P3.5) by (F1) and (F2) with $v_{31} = v_1$ (F3.5) and $v_{32} = v_2$ (F3.6). Having proven all premises (P3.1-4) from (IH3.1) we obtain:

$$\begin{aligned}
& (v_{31}, v_{32}) \in \mathcal{V}[\llbracket \tau \rrbracket^{\rho_3} \\
& \equiv \langle \text{By (F3.5-6)} \rangle \\
& (v_1, v_2) \in \mathcal{V}[\llbracket \tau \rrbracket^{\rho_3} \\
& \equiv \langle \text{By Lemma B.8 knowing that } i_1, i_2 \notin \Psi.d \text{ and } j \notin \Psi.l \rangle \\
& (v_1, v_2) \in \mathcal{V}[\llbracket \tau \rrbracket^{\rho_3 \setminus \{i_1, i_2, j\}} \\
& \equiv \langle \text{By def of } \rho_3 \text{ and } (_ \setminus _) \rangle \\
& (v_1, v_2) \in \mathcal{V}[\llbracket \tau \rrbracket^\rho
\end{aligned}$$

Which is precisely our goal (C').

□

Bibliography

- [1] Combining deep and shallow embedding of domain-specific languages. *Comput. Lang. Syst. Struct.*, dec 2015.
- [2] C. Abuah, D. Darais, and J. P. Near. Solo: Enforcing differential privacy without fancy types. *CoRR*, abs/2105.01632, 2021.
- [3] C. Abuah, A. Silence, D. Darais, and J. P. Near. DDUO: general-purpose dynamic analysis for differential privacy. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–15. IEEE, 2021.
- [4] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *Proceedings of the 15th European Conference on Programming Languages and Systems, ESOP’06*. Springer-Verlag, 2006.
- [5] M. Algehed and J.-P. Bernardy. Simple noninterference from parametricity. *Proc. ACM Program. Lang.*, 2019.
- [6] G. Barthe, G. P. Farina, M. Gaboardi, E. J. G. Arias, A. Gordon, J. Hsu, and P. Strub. Differentially private bayesian programming. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 68–79, 2016.
- [7] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, A. Roth, and P.-Y. Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. In *POPL’15*. ACM, 2015.
- [8] J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. *SIGPLAN Not.*, 2010.
- [9] J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free: Parametricity for dependent types. *Journal of Functional Programming*, 22:107–152, 03 2012.
- [10] J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *Proc. Annual IEEE/ACM Symposium on Logic in Computer Science, LICS ’12*. IEEE Computer Society, 2012.
- [11] W. J. Bowman and A. Ahmed. Noninterference for free. *SIGPLAN Not.*, 2015.
- [12] S. Chaudhuri, S. Gulwani, and R. Lubliner. Continuity and robustness of programs. *Communications of the ACM*, 55(8):107–115, 2012.
- [13] L. Cruz-Filipe, H. Geuvers, and F. Wiedijk. In *MKM, volume 3119 Lecture Notes in Computer Science*. Springer, 2004.

- [14] L. D’Antoni, M. Gaboardi, E. J. G. Arias, A. Haeberlen, and B. C. Pierce. Sensitivity analysis using type-based constraints. In R. Lazarus, A. J. Kfoury, and J. Beal, editors, *Proceedings of the 1st annual workshop on Functional programming concepts in domain-specific languages, FPCDSL@ICFP 2013, Boston, Massachusetts, USA, September 22, 2013*, pages 43–50. ACM, 2013.
- [15] A. A. de Amorim, M. Gaboardi, J. Hsu, and S. Katsumata. Probabilistic relational reasoning via metrics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24–27, 2019*, pages 1–19. IEEE, 2019.
- [16] A. A. de Amorim, M. Gaboardi, J. Hsu, S. Katsumata, and I. Cherigui. A semantic account of metric preservation. In G. Castagna and A. D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18–20, 2017*, pages 545–556. ACM, 2017.
- [17] T. C. development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.
- [18] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography, TCC’06*, pages 265–284, 2006.
- [19] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
- [20] F. Eigner and M. Maffei. Differential privacy by typing in security protocols. In *2013 IEEE 26th Computer Security Foundations Symposium, New Orleans, LA, USA, June 26–28, 2013*, pages 272–286. IEEE Computer Society, 2013.
- [21] M. Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.
- [22] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2013.
- [23] H. Geuvers and M. Niqui. Constructive reals in coq: Axioms and categoricity. In *Types for Proofs and Programs, International Workshop, TYPES 2000, Selected Papers*, Lecture Notes in Computer Science. Springer, 2000.
- [24] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *Proc. of USENIX Security Symposium*, 2011.
- [25] N. M. Johnson, J. P. Near, J. M. Hellerstein, and D. Song. Chorus: a programming framework for building scalable differential privacy mechanisms. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7–11, 2020*, pages 535–551. IEEE, 2020.

- [26] M. P. Jones. A theory of qualified types. *Sci. Comput. Program.*, 22(3), 1994.
- [27] E. Lobo-Vesga, A. Russo, and M. Gaboardi. A programming framework for differential privacy with accuracy concentration bounds. In *Proc. IEEE Symp. on Security and Privacy*, SP '20. IEEE Computer Society, 2020.
- [28] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*. ACM, 2009.
- [29] J. P. Near, D. Darais, C. Abua, T. Stevens, P. Gaddamadugu, L. Wang, N. Soman, M. Zhang, N. Sharma, A. Shan, and D. Song. Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy. *Proc. ACM Program. Lang.*, 3(OOPSLA):172:1–172:30, 2019.
- [30] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.
- [31] D. Otmani and R. A. Eisenberg. The thoral plugin: For your fancy type needs. In *Proc. of the 11th ACM SIGPLAN International Symposium on Haskell*, Haskell. ACM, 2018.
- [32] M. Pickering, G. Érdi, S. Peyton Jones, and R. A. Eisenberg. Pattern synonyms. *SIGPLAN Not.*, 2016.
- [33] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *Proc. ACM SIGPLAN International Conference on Functional Programming*, 2010.
- [34] J. C. Reynolds. Types, Abstraction, and Parametric Polymorphism. In *Information Processing 83: Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983*. Elsevier Science Publishers B. V. (North-Holland), 1983.
- [35] P. M. Rondon, M. Kawaguchi, and R. Jhala. Liquid types. In *Proc. of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation*. ACM, 2008.
- [36] D. Terei, S. Marlow, S. L. Peyton Jones, and D. Mazières. Safe Haskell. In *Proceedings of the 5th ACM SIGPLAN Symposium on Haskell, Haskell 2012, Copenhagen, Denmark, 13 September 2012*, pages 137–148, 2012.
- [37] M. Toro, D. Darais, C. Abua, J. Near, F. Olmedo, and É. Tanter. Contextual linear types for differential privacy. *CoRR*, abs/2010.11342, 2020.
- [38] N. Vazou, E. L. Seidel, and R. Jhala. LiquidHaskell: experience with refinement types in the real world. In *Proc/ of the 2014 ACM SIGPLAN symposium on Haskell*. ACM, 2014.

- [39] P. Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 347–359, 1989.
- [40] P. Wadler and S. Blott. How to make ad-hoc polymorphism less ad-hoc. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages*, 1989.
- [41] Y. Wang, Z. Ding, D. Kifer, and D. Zhang. Checkdp: An automated and integrated approach for proving differential privacy or finding precise counterexamples. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9–13, 2020*, pages 919–938. ACM, 2020.
- [42] Y. Wang, Z. Ding, G. Wang, D. Kifer, and D. Zhang. Proving differential privacy with shadow execution. In K. S. McKinley and K. Fisher, editors, *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22–26, 2019*, pages 655–669. ACM, 2019.
- [43] D. Winograd-Cort, A. Haeberlen, A. Roth, and B. C. Pierce. A framework for adaptive differential privacy. *PACMPL*, 1(ICFP):10:1–10:29, 2017.
- [44] D. Zhang and D. Kifer. LightDP: towards automating differential privacy proofs. In *Proc. ACM SIGPLAN Symp. on Principles of Programming Languages*, 2017.
- [45] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. EKTELO: A framework for defining differentially-private computations. In *Proc. International Conference on Management of Data*, 2018.
- [46] H. Zhang, E. Roth, A. Haeberlen, B. C. Pierce, and A. Roth. Fuzzi: a three-level logic for differential privacy. *Proc. ACM Program. Lang.*, 3(ICFP):93:1–93:28, 2019.