Generalized Policy-Based Noninterference for Efficient Confidentiality-Preservation

SHAMIEK MANGIPUDI, Università della Svizzera italiana (USI), Switzerland
PAVEL CHUPRIKOV, Università della Svizzera italiana (USI), Switzerland
PATRICK EUGSTER, Università della Svizzera italiana (USI), Switzerland
MALTE VIERING, TU Darmstadt, Germany
SAVVAS SAVVIDES, Purdue University, USA

ABSTRACT

As more organizations are leveraging third-party cloud and edge data centers to process data efficiently, the issue of preserving data confidentiality becomes increasingly important. In response, numerous security mechanisms have been introduced and promoted in recent years including software-based ones such as homomorphic encryption, as well as hardware-based ones such as Intel SGX and AMD SEV. However these mechanisms vary in their security properties, performance characteristics, availability, and application modalities, making it hard for programmers to judiciously choose and correctly employ the right one for a given data query.

This paper presents a mechanism-independent approach to distributed confidentiality-preserving data analytics. Our approach hinges on a core programming language which abstracts the intricacies of individual security mechanisms. Data is labeled using custom confidentiality levels arranged along a lattice in order to capture its exact confidentiality constraints. High-level mappings between available mechanisms and these labels are captured through a novel expressive form of security policy. Confidentiality is guaranteed through a type system based on a novel formulation of noninterference, generalized to support our security policy definition. Queries written in a largely security-agnostic subset of our language are transformed to the full language to automatically use mechanisms in an efficient, possibly combined manner, while provably preserving confidentiality in data queries end-to-end. We prototype our approach as an extension to the popular Apache Spark analytics engine, demonstrating the significant versatility and performance benefits of our approach over single hardwired mechanisms — including in existing systems — without compromising on confidentiality.

CCS Concepts: • **Security and privacy → Information flow control**; **Distributed systems security**; **Trusted computing**; • **Computer systems organization → Cloud computing**.

Additional Key Words and Phrases: language-based security, type system, noninterference, enclave, secure computing, homomorphic encryption

ACM Reference Format:
Shamiek Mangipudi, Pavel Chuprikov, Patrick Eugster, Malte Viering, and Savvas Savvides. 2023. Generalized Policy-Based Noninterference for Efficient Confidentiality-Preservation. *Proc. ACM Program. Lang.* 7, PLDI, Article 117 (June 2023), 61 pages. https://doi.org/10.1145/3591231

Authors' addresses: Shamiek Mangipudi, Università della Svizzera italiana (USI), Lugano, Switzerland, mangish@usi.ch; Pavel Chuprikov, Università della Svizzera italiana (USI), Lugano, Switzerland, pavel.chuprikov@usi.ch; Patrick Eugster, Università della Svizzera italiana (USI), Lugano, Switzerland, eugstp@usi.ch; Malte Viering, TU Darmstadt, Darmstadt, Germany, viering@dsp.tu-darmstadt.de; Savvas Savvides, Purdue University, West Lafayette, USA, savvas@purdue.edu.

1 INTRODUCTION

A primary challenge when leveraging cost-effective third-party cloud and edge data centers for processing data is to ensure that data confidentiality constraints are satisfied. To that end several confidentiality-preserving mechanisms have been proposed, differing in their security properties and performance characteristics, as well as where/how available they are and how they are used.

Options for secure data processing in third-party clouds. *Homomorphic encryption* (HE) allows data to be computed with while it is encrypted. *Fully homomorphic encryption* (FHE) cryptosystems [Gentry 2009] support arbitrary computations over encrypted data but exhibit substantial overheads. *Partially homomorphic encryption* (PHE) cryptosystems such as Paillier [1999] or ElGamal [1985] are generally much more efficient, however, each cryptosystem can only support certain operations – addition or multiplication of ciphertexts respectively in the above two cases. These limitations can be overcome, e.g., by using few trusted resources on the client side to complete queries by performing remaining operations on data in plaintext [Tu et al. 2013] or re-encrypting data [Stephen et al. 2014a]. But determining when and how to do so most efficiently in a given data query adds to the difficulty of choosing between different cryptosystems, understanding their security properties, and employing them correctly in combination.

Several hardware-based mechanisms have also been proposed, including Intel's *software guard extensions* (SGX), AMD's *secure encrypted virtualization* (SEV) etc. Besides not being ubiquitously available (e.g., Microsoft Azure provides SGX while Google GCP provides SEV), these each have specific security and performance properties, and are non-trivial to set up (e.g., using remote attestation via trust authority) and use by programmers (e.g., identify sensitive data, reason about information flow, partition programs to minimize trusted computing base).

A flexible mechanism-independent approach. To help data analysts without expert knowledge in security efficiently query data using third-party compute resources without compromising data confidentiality, we present a novel mechanism-independent approach that employs an extensible set of software- and hardware-based security mechanisms for expressing confidentiality-preserving computations. In short, our approach hinges on a novel general form of *security policy* $\mathbb{S}$, and a corresponding novel formulation of the theory of *noninterference* (NI) [Goguen and Meseguer 1982] for our general notion of security policy: $\mathbb{S}$-*noninterference* ($\mathbb{S}$-NI) .

That is, as security mechanisms provide different properties, not all data usually has the same confidentiality constraints, and stronger constraints tend to call for more costly mechanisms, our model allows (1) different custom levels of data confidentiality in the form of labels arranged along a lattice following established practices (e.g., [Denning 1976; Denning and Denning 1977; Sandhu 1993]). (2) Data sets are assigned labels of the lattice to capture their confidentiality constraints in a fine-grained manner. (3) Our novel security policy associates the labels of the lattice with both different cryptosystems (schemes) and available hardware mechanisms (domains). Finally, (4) data queries expressed in a programming language marrying functions, relations, and query operators are executed efficiently in a way leveraging the different mechanisms individually or in combination using annotations to meet constraints on the data's confidentiality and on mechanisms as captured by the security policy.

To help harness our model we develop a type system to statically ensure secure use of mechanisms, based on our novel theory of $\mathbb{S}$-NI . Furthermore, we formalize the process of correct transformation allowing data analysts to express queries in a subset of our language without security annotations, which are then transformed to use security mechanisms based on (3) such as to guarantee data confidentiality constraints as per (2).

We describe a prototype implementation of our approach dubbed HYDRA (hybrid approach to distributed confidentiality-preserving data analytics) based on the popular Apache Spark [Zaharia et al. 2012] data analytics platform. In our prototype (cf. Fig. 1), programmers specify queries in Scala using Spark SQL [Armbrust et al. 2015] which are then augmented with annotations for efficient execution with confidentiality preserved end-to-end using one of several current heuristics: (a) "PHE only" using PHE cryptosystems, and client-side completion when encountering limits of PHE; (b) "SGX only" using exclusively SGX; (c) a simple hybrid heuristic combining PHE and SGX based on a cost model of individual operations. Performance evaluation on the popular TPC-H benchmark [TPC 1988] in Amazon AWS demonstrates that (a) and (b) are competitive with respect to state-of-the art solutions Cuttlefish [Savvides et al. 2017] and Opaque [Zheng et al. 2017] respectively, which are also based on Apache Spark but with hardwired mechanisms (in fact HYDRA is on average substantially faster – 1.6× and 11.3× respectively). We also show that (c) commonly outperforms (a) and (b), on average by a significant 1.7× and 1.6× respectively, demonstrating the benefits not only of supporting different mechanisms to make queries more portable, but of allowing mechanisms to be combined in a secure manner for improved performance.

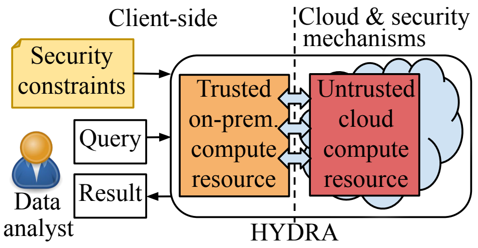
Client-side
Cloud & security mechanisms
Security constraints
Query
Result
Data analyst
Trusted on-prem. compute resource
Untrusted cloud compute resource
HYDRA

Fig. 1. HYDRA overview. Data analysts submit queries to be executed using shared cloud resources (and, if need be, limited local resources), and collect results. Queries are transformed based on security constraints to best use available security mechanisms for confidentiality preservation.

Contributions and roadmap. In summary, this paper makes the following contributions:

- A core programming language for confidentiality-preserving computation atop third-party compute infrastructures which allows to combine security mechanisms.
- A type system for our language to statically ensure correct, confidentiality-preserving combination among different mechanisms based on a novel general formulation of NI dubbed $\mathbb{S}$-NI which combines domains, schemes, and labels via a general security policy $\mathbb{S}$, and a formal transformation process to augment queries with annotations for secure and correct use of security mechanisms. On top of traditional information flow control, our type system tackles extra challenges stemming from the combined use of two orthogonal (hence, incomparable) kinds of security mechanisms, namely domains and schemes, and presents a more compact treatment of the implicit effect relation sizes have on computation's result.
- A prototype implementation of our approach for the Spark data analytics engine along with three heuristics to respectively leverage PHE (with client-side completion), SGX, or a combination of the two.
- An evaluation of our prototype using our three heuristics on an industrial benchmark showing the benefits of being able to switch between different security mechanisms and also of combining them.

Note that for simplicity we use the term HYDRA in the following to refer to our approach and its underlying model and language, in addition to the prototype. The meaning is clear from context.

The rest of this paper is organized as follows. § 2 gives background information and introduces the lattice-based model of confidentiality used in HYDRA with examples and HYDRA's workflow. § 3 introduces the static and dynamic semantics of HYDRA's core language, and § 4 presents a type system for reasoning about confidentiality end-to-end, corresponding formal properties, and a formalization of the query transformation process. § 5 discusses HYDRA's implementation in Spark.

§ 6 evaluates our approach. § 7 contrasts Hydra with related work, and § 8 concludes with final remarks.

Detailed proofs of all lemmas and theorems can be found in the appendix with additional definitions.

2 OVERVIEW OF Hydra

Hydra achieves its confidentiality goals by using "pluggable" *(execution) domains* and *(encryption) schemes*. We denote the set of domains and schemes as $\mathcal{D}$ and $\mathcal{S}$ respectively. As an example, $\mathcal{D}$ may include public or private cloud, trusted client side, or SGX. Data always resides in some domain, but is not always encrypted, hence for uniformity we would also use $\mathcal{S}_\varnothing = \mathcal{S} \cup \{\varnothing\}$, where $\varnothing$ means plaintext (no encryption scheme). We impose a partial order on $\mathcal{D} \times \mathcal{S}_\varnothing$ representing "less secure or equal", namely $(d, \varnothing) \preccurlyeq_{ds} (d, s)$ for any $s \in \mathcal{S}_\varnothing$. Each element of $\mathcal{D} \times \mathcal{S}_\varnothing$ provides a certain confidentiality level irrespective of what the underlying plaintext data is. Similarly, every data item comes with a confidentiality level requirement, realizable in different ways.

2.1 Threat Model

We consider *honest but curious* (HbC) adversaries [Dong et al. 2016, 2018; Popa et al. 2012; Savvides et al. 2017; Tople et al. 2013], that can see the data, but cannot modify it. Adversaries differ in what they can see. Each adversary A is represented by a *downward-closed* set (w.r.t $\preccurlyeq_{ds}$) $A \subseteq \mathcal{D} \times \mathcal{S}_\varnothing$ of domains and schemes that one is able to *break*, i.e., able to observe the protected data. The downward-closed constraint captures the fact that if A is not able to observe plaintext in some domain, i.e., $(d, \varnothing) \notin A$, then surely A is not able to observe any (no less secure) ciphertext either, i.e., for every $s \in \mathcal{S}$, $(d, s) \notin A$ since $(d, \varnothing) \preccurlyeq_{ds} (d, s)$. That observation is valid irrespective of any specific $\mathcal{D}$ and $\mathcal{S}$. We denote the set of all such adversaries as $\mathcal{A}$ and also regard it as a partially ordered set $\langle \mathcal{A}, \subseteq \rangle$. Intuitively, $A \subseteq A'$ is the same as "A is no more powerful than A'". We assume that each adversary A can only observe the *final result* of the computation, and that observation is *partial* based on domain/scheme combinations A is able to break (elements of A).

Via these generic adversaries, we abstract away computational guarantees of encryption schemes and trustworthiness of domains and leave it to a *security expert* to decide which combinations suffice for a given confidentiality level (using abstractions described shortly in § 2.2). Viaduct [Acay et al. 2021] used a similar approach to represent adversaries, associating each with a set of hosts (or principals) it can read data from. Unlike Hydra, Viaduct does not have a general treatment of encryption schemes: it presents four specific protocols able to alter the set of hosts reading or writing data with some restrictions on protocol composition. Attacks on data integrity and availability are out of Hydra's scope. Even though we do not consider specific side-channel attacks, Hydra's customizable security policy allows to take into account security mechanisms with known side-channel and access pattern attacks, and appropriately preclude usage of such mechanisms for highly sensitive confidentiality levels.

Given the security policy, our system prevents *direct information leaks*, and *indirect information leaks* except those via size of the query result. The former represent direct violations of the policy. The latter represent gaining *some* knowledge about one part of the data (e.g. secret) by observing a different part of the data (e.g. non-secret), e.g., gaining information about the inputs (e.g. secret) to a filter's predicate based on other (e.g. non-secret) columns in the result, or about the key (e.g. secret) used for aggregation based on the aggregated values (e.g. non-secret).

Preventing indirect leaks via results size would be too restrictive (e.g., no PHE-based filtering in the public cloud), and padding, typically done to prevent size leaks, incurs substantial overhead.

2.2 Security Constraints

Confidentiality constraints in Hydra are based on a finite set $\mathcal{L}$ of *security labels* (also levels or security classes, cf. [Denning 1976; Denning and Denning 1977; Sandhu 1993]) reflecting differences in confidentiality requirements of the data. $\mathcal{L}$ is typically defined by an organization's *business expert* together with a *security expert*. The latter entity also defines the *security policy*: correspondence between the confidentiality requirements represented by a given security label and the set of execution domains $\mathcal{D}$ and encryption schemes $\mathcal{S}$ satisfying those requirements. A *data manager* then typically provides database schemata where fields (columns) of relations are annotated with labels from $\mathcal{L}$. The labels restrict, indirectly through a security policy, a set of adversaries who must not be able to get access to the corresponding data. Security labels in $\mathcal{L}$ thus serve as abstractions for: (1) confidentiality requirements of data columns, and (2) confidentiality guarantees accorded by schemes (software-based security mechanisms) and domains (hardware-based security mechanisms or trusted execution environments).

2.2.1 Security Lattice. To entertain combining different confidentiality levels in the course of computation, Hydra demands $\mathcal{L}$ to be equipped with a lattice structure following Denning [1976], thus forming a *security lattice* $\langle \mathcal{L}, \preccurlyeq, \sqcup, \sqcap \rangle$, where (join) $\sqcup : \mathcal{L} \times \mathcal{L} \longrightarrow \mathcal{L}$ is the least upper bound operator, (meet) $\sqcap : \mathcal{L} \times \mathcal{L} \longrightarrow \mathcal{L}$ is the greatest lower bound operator, and (partial order) $\preccurlyeq$ is a reflexive, transitive, and antisymmetric binary relation on $\mathcal{L}$. The join $l_1 \sqcup l_2$ is at least as strict as both l_1 and l_2; it is used, in particular, to label the result of an operation involving two differently labelled inputs. The meet $l_1 \sqcap l_2$ is at least as lenient as both l_1 and l_2. If $l_1 \preccurlyeq l_2$ then confidentiality requirements imposed by l_2 are at least as strict as those imposed by l_1, and it is always safe to replace l_1 with l_2 when labelling data. As $\mathcal{L}$ is finite, there exists the lowest security label $\bot \in \mathcal{L}$, such that, $\bot \preccurlyeq l$ for all $l \in \mathcal{L}$; label $\bot$ represents data with no confidentiality constraints. There is a conceptual difference between $\preccurlyeq_{ds}$ and $\preccurlyeq$: the former captures an intrinsic property of adversaries that is independent of specific $\mathcal{D}$, $\mathcal{S}$ and $\mathcal{L}$; the latter compares confidentiality levels, but ultimately does not change the threat model, i.e., the set $\mathcal{A}$. On a technical side, $\preccurlyeq_{ds}$ determines the partial ordering on elements (d, s) of adversary A while $\preccurlyeq$ determines partial ordering on labels $l \in \mathcal{L}$.

2.2.2 Security Policy. To inform Hydra about confidentiality guarantees provided by domains and schemes, Hydra uses a novel generalized *security policy* represented as a function $\mathbb{S} : \mathcal{L} \longrightarrow 2^{\mathcal{D} \times \mathcal{S}_\varnothing}$, such that for every $l \in \mathcal{L}$ we have $\mathbb{S}(l)^c \in \mathcal{A}$ and $-^c \circ \mathbb{S} : \mathcal{L} \longrightarrow \mathcal{A}$ is *order- and minimum-preserving*; we use X^c to denote the set complement of X, e.g., $X^c = (\mathcal{D} \times \mathcal{S}_\varnothing \setminus X)$ for $X \subseteq \mathcal{D} \times \mathcal{S}_\varnothing$.

The intuitive interpretation of the security policy is that each element of $\mathbb{S}(l)$ provides confidentiality guarantees sufficient for level l. Formally, any adversary A *unable* to observe values once protected by *any* combination from $\mathbb{S}(l)$, $A \cap \mathbb{S}(l) = \emptyset$, must *not* be able to access inputs labeled by l. Equivalently, $\mathbb{S}(l)^c$ denotes the most powerful adversary who must *not* be able to access inputs labelled by l, i.e., $A \in \mathcal{A}$ must *not* be able to access l-inputs if and only if $A \subseteq \mathbb{S}(l)^c$. In light of that interpretation, preservation of the minimum, $\mathbb{S}(\bot)^c = \emptyset$, ensures confidentiality level $\bot$ has the intended meaning of "no confidentiality constraints" so the most powerful adversary who must *not* be able to access $\bot$ is the one unable to observe anything; order preservation says that if $l' \preccurlyeq l$ then any adversary who must not be able to access l'-labelled inputs must not be able to access l-labelled inputs.

When expressing the policy in a tabular form (e.g., Tbl. 1) we assume that each occurrence of (d, s, l) in the same row is representing $(d, s) \in \mathbb{S}(l)$.

2.2.3 Example. Fig. 2 shows a simple example of a security lattice with $\mathcal{L} = \{\texttt{Public}, \texttt{Low}, \texttt{High}\}$ and associated security policy used in other works (e.g., [Gollamudi and Chong 2016]) and for

Table 1. A relation producing a security policy for the simple lattice of Fig. 2. For brevity we elide some triples of security policy inferrable from $-^c \circ \mathbb{S} : \mathcal{L} \to \mathcal{A}$ being order- and minimum-preserving.

Label $\mathcal{L}$	Domain $\mathcal{D}$	Scheme $\mathcal{S}_\varnothing$
`High`	`CLNT`, `SGX`	$\varnothing$
`High`	`CLD`	`AES-GCM`
`Low`	`CLD`	`SWP`, `AES-ECB`, `Paillier`, `ElGamal`, `OPE`

Table 2. Extract of `Customers` and `Orders` relations of the TPC-H benchmark. Labels are from the simple lattice shown with Fig. 2. Security types are for execution in `CLD` and are inferred based on the security policy from Tbl. 1 and on the operations used by the example query of Fig. 5.

Relation	Field	Type	Label	Security type (inferred)
`Customers`	`custId`	`Str`	`Low`	$(\texttt{Str}^{\texttt{AES-ECB}},\texttt{Low})$
	`bal`	`Dbl`	`High`	$(\texttt{Dbl}^{\texttt{AES-GCM}},\texttt{High})$
	...			
`Orders`	`orderId`	`Str`	`High`	$(\texttt{Str}^{\texttt{AES-GCM}},\texttt{High})$
	`custKey`	`Str`	`Low`	$(\texttt{Str}^{\texttt{AES-ECB}},\texttt{Low})$
	`price`	`Str`	`High`	$(\texttt{Dbl}^{\texttt{AES-GCM}},\texttt{High})$
	`date`	`Int`	`Low`	$(\texttt{Int}^{\texttt{OPE}},\texttt{Low})$
	...			

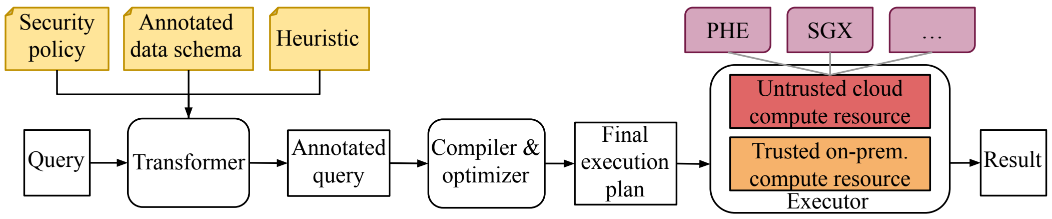

Fig. 3. HYDRA workflow. Except for red parts execution happens on the client side where the analyst resides.

the TPC-H benchmark in our evaluation later. Domain `CLNT` means (trusted) *client side*, and `CLD` public (untrusted) cloud. Triples from the first two rows in Tbl. 1 state that data labeled `High` is allowed to be in plaintext when present inside domains `CLNT` and `SGX`, and must be encrypted under `AES-GCM` when present inside `CLD`. The third row states that data labeled `Low` should be encrypted under any of the five listed schemes when inside `CLD`. Note, as $\mathbb{S}(l)^c \in \mathcal{A}$, $\mathbb{S}(l)$ must be upward closed, and since having `High`-sensitive in plaintext inside `SGX` is secure (according to Tbl. 1), $(\texttt{SGX}, \varnothing) \in \mathbb{S}(\texttt{High})$, then it is surely secure to have that same data inside `SGX` encrypted under any encryption scheme s. An example of a more complex lattice is discussed in the appendix. Tbl. 2 shows an extract of the correspondingly annotated TPC-H data set. Fields such as `bal`(ance) containing highly sensitive data that are annotated with a `High` label are (as inferred by HYDRA) encrypted with *advanced encryption standard Galois counter mode* (AES-GCM) while those such as `date` containing moderately sensitive data are annotated with `Low` are encrypted with *advanced encryption standard electronic codebook* (AES-ECB) or *order-preserving encryption* (OPE).

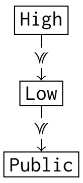

Fig. 2. Simple lattice.

2.2.4 Escape Hatch. Note that our approach currently does not include a primitive for declassification, as it already has the inherent escape hatch of running on the trusted client side when out of options to continue execution in the cloud. Label creep — an issue stemming from NI being too strong — is less of an issue in our approach since all data whose labels crawl to the highest level are processed inside the most secure domain (e.g., `SGX`), but the computed result can still be encrypted and remain in the cloud until returned to the trusted client side for decryption.

2.3 HYDRA Workflow

Fig. 3 shows the end-to-end workflow of HYDRA for a given query once the security policy is set up and data has been labeled. Apart from the untrusted (third-party shared) cloud compute

$$
\begin{array}{rrcl}
\text{Type} & \kappa & ::= & \overline{\kappa} \rightarrow_{d} \kappa \mid \{\overline{f : (p^{s}, l)}\} \mid T\{\overline{f : (p^{s}, l)}\} \mid (p^{s}, l) \\
\text{Prim type} & p & ::= & \texttt{Int} \mid \texttt{Dbl} \mid \texttt{Str} \mid \texttt{Bool} \mid \ldots \\
\text{Scheme} & s & ::= & \varnothing \mid \texttt{AES-GCM} \mid \texttt{ElGamal} \mid \texttt{Paillier} \mid \ldots \quad \text{(in one-to-one correspondence with } \mathcal{S}_{\varnothing}) \\
\text{Domain} & d & ::= & \texttt{CLNT} \mid \texttt{SGX} \mid \texttt{SEV} \mid \texttt{CLD} \mid \ldots \quad \text{(in one-to-one correspondence with } \mathcal{D}) \\
\text{Value} & v & ::= & T\{\overline{f : v}\} \mid \{\overline{f : v}\} \mid c^{s} \mid \lambda[d](\overline{x : \kappa}).\ e \mid f \\
\text{Expression} & e & ::= & v \mid x \mid e(\overline{e}) \mid \oplus(\overline{e}) \mid \{\overline{f : e}\} \mid e.f \mid \texttt{table}(name) \mid \theta(\overline{e}) \mid \texttt{encr}(e, s) \mid \texttt{decr}(e) \mid [e]_{d} \\
\text{Prim ops} & \oplus & ::= & + \mid - \mid \ldots \mid \wedge \mid \vee \mid \ldots \\
\text{Query ops} & \theta & ::= & \texttt{filter} \mid \texttt{proj} \mid \texttt{cross} \mid \texttt{agg} \mid \ldots
\end{array}
$$

Fig. 4. Syntax and parameterization of Hydra language. Terms/items in *green* (only present at runtime), and *blue* are not used by the data analyst. The superscript s in the base type p^s denotes either a plaintext ($s = \varnothing$) or encrypted ($s \in \mathcal{S}$) version of primitive type p. An overline represents a sequence.

resources, execution occurs at the trusted client side (where the data analyst resides, cf. Fig. 3). Data is stored beforehand in encrypted form, as needed, in the cloud.

Transformer: The "logical" query without security annotations submitted by the data analyst is transformed to use security mechanisms based on the annotated data schema (with labels), the security policy, and a heuristic for using mechanisms defined in the security policy.

Compiler & optimizer: The compiler takes a query with explicit use of mechanisms and annotations for security, verifies it, and generates a final optimized execution plan.

Executor: The execution back-end uses untrusted third-party resources with different mechanisms, cryptosystems, and client-side trusted resources (if needed) to perform the query on the data in the cloud and generate the (encrypted) result, which is sent to the data analyst.

The workflow steps are extensible in several ways: a more sophisticated algorithm for mechanism selection means extending the *transformer*; a new encryption scheme amounts to adding several Spark Catalyst rules (see § 5.2) to the *compiler* and making the *transformer* aware of the scheme's performance; a new domain includes the steps for a new encryption scheme plus telling the *executor* how to execute subquery in that domain. As input queries have no security annotations and intermediate steps are *not* persisted, the executed query is always compatible with the runtime.

3 PROGRAMMING LANGUAGE

We present syntax and operational semantics of Hydra's core programming language.

3.1 Syntax

The syntax, shown in Fig. 4 , reflects three aspects of confidentiality-preserving distributed data processing: (composition of) query operators applied to relational data, (user-defined) functions that parameterize query operators, and confidentiality constraints. $\overline{z}$ is a sequence $z_1, \ldots, z_n$. In some cases (not sequences) we may also use z', z'' etc. to range over several instances of a meta-variable. In our prototype, the data analyst only uses an abbreviated subset of the language without security annotations (*blue*) and runtime-only constructs (*green*).

3.1.1 Values. Values v include two constructs central to data representation, namely, relations $T\{\overline{f : v}\}$ and records of values $\{\overline{f : v}\}$, where we abbreviate $\overline{f : v} = f_1 : (v_{11}, \ldots, v_{k1}), \ldots, f_n : (v_{1n}, \ldots, v_{kn})$ and $\overline{y : z} = y_1 : z_1, \ldots, y_n : z_n$. Other values of our language are: possibly encrypted constants c^s, where $s \in \mathcal{S}_\varnothing$ and c is an element of a ground set of primitive values (e.g., an integer `42`, a string `"hello"`, or a sequence of bytes `0x42be`... representing some ciphertext); lambda abstractions $\lambda[d](\overline{x : \kappa}).\ e$; and, as a minor technical convenience, record fields f. Notably, every lambda abstraction explicitly states *domain* d in which its body must be evaluated.

```
1 agg(filter(cross(table(Customers),
2                  filter(table(Orders),
3                         λ(rO: {/* Orders */}). rO.date < 16052002)),
4             λ(rCO: {/* Customers + Orders */}). rCO.custId == rCO.custKey),
5     custId, 0,
6     λ(rP: {price: Dbl}, acc: Int).
7         acc + rP.price)
```

Fig. 5. Example query without security annotations as expressed by a data analyst. The types of record fields inside comments `/* ... */` are taken from the "Type" column of Tbl. 2.

3.1.2 Expressions. Besides values v, expressions e include several primitives standard for lambda calculus with records and primitive data types such as variables x, function applications $e(\overline{e})$, primitive arithmetic and logical operations denoted by $\oplus(\overline{e})$, records $\{\overline{f : e}\}$, and record access $e.f$. When an operator $\oplus$ is binary, we will use infix notation $e_1 \oplus e_2$ to mean $\oplus(e_1, e_2)$. Then, there are two data-processing constructs: references to relations $\texttt{table}(name)$ already in the database, and relational query operators $\theta(\ldots)$ such as `filter` or `cross` that transform relations. Encryption and decryption primitives are represented by $\texttt{encr}(e, s)$ and $\texttt{decr}(e)$. In $\texttt{encr}(e, s)$, $s \neq \varnothing$ defines the encryption scheme; it is implicit in $\texttt{decr}(e)$. Neither `encr` nor `decr` contain keys, we provide keys at runtime to a well-typed query appropriately, assuming one encryption key per encryption scheme. Finally, during runtime we use $[e]_d$ to demarcate a sub-expression evaluated inside domain d.

3.1.3 Types. Types are parameterized by user-provided lattice $\mathcal{L}$. At the lowest level we have *primitive types* p, including integer and floating point numbers, and strings. The *base types* p^s, $s \in \mathcal{S}_\varnothing$ are for optionally encrypted values of primitive types. By attaching security labels $l \in \mathcal{L}$ and structure to base types, we get the following *types* κ: (p^s, l) for atomic values, $\{\overline{f : (p^s, l)}\}$ for records, and $T\{\overline{f : (p^s, l)}\}$ for sequences of records, i.e., relational schemata; $\overline{f : (p^s, l)}$ stands for $f_1 : (p_1^{s_1}, l_1), \ldots, f_n : (p_n^{s_n}, l_n)$. Finally, we have a function type $\overline{\kappa} \rightarrow_d \kappa$ carrying a domain d inside which the function will execute, in addition to usual parameter and result types. The domain may represent Intel SGX (SGX), client-side computation (CLNT), AMD SEV (SEV), the public cloud without hardware security mechanism (CLD), etc. To capture structure of inputs, we use *table environment* $\rho : name \mapsto T\{\overline{f : (p^s, l)}\}$, a *finite* map from relations' names to their schemata. The map is provided by the data manager (see § 2.2) without encryption schemes, i.e., all $s = \varnothing$; the encryption schemes are inferred by query transformation (see § 3.3).

3.1.4 Syntactic Sugar For the sake of simplicity, in the examples, we use a shorthand notation without security annotations, the same notation that is used by a data analyst when writing queries. The simplified notation boils down to replacing parts colored in *blue* with some defaults; we remind that *green* expressions exist only at runtime. The defaults are the following: an omitted domain is replaced with the client-side domain CLNT, an omitted scheme s is replaced with $\varnothing$, and an omitted security label is replaced with $\bot$. As an example, $\lambda(x : \texttt{Int}).\ x + 2$ desugars to $\lambda[\texttt{CLNT}](x : (\texttt{Int}^\varnothing, \bot)).\ x + 2^\varnothing$. The example in Fig. 5 is expressed in the simplified notation and hence the type annotations correspond to the non-colored portions of κ defined in Fig. 4.

3.1.5 Running Example. Fig. 5 presents a simple query retrieving the amount of money spent by each customer (see `Customers` in Tbl. 2) on orders (see `Orders` in Tbl. 2) prior to a certain date. The query has three steps: (1) filter orders of interests on line 2, (2) perform an inner join with customers using `cross` followed by `filter` on lines 1–4, and (3) aggregate `price` by `cusutId` using `agg`.

3.1.6 Primitives. Hydra makes extensible the set $\mathcal{S}$ of possible schemes, domains $\mathcal{D}$, primitive types p, and operations $\oplus$ (see Fig. 4). Semantics of primitive operations is captured by $\varphi_\oplus^{\mathrm{ev}}(\oplus, \overline{c, s})$, a partial map from syntax symbols (e.g., + or -) and primitive values with possibly encryption schemes (cf. (Ev Op)) to a primitive result value and possibly scheme; $\overline{z, s}$ stands for $z_1, s_1, \ldots, z_n, s_n$.

Table 3. Functions capturing HYDRA's built-in types ($\varphi^{\mathsf{ty}}_{\_}$) and operations ($\varphi^{\mathsf{ev}}_{\_}$) with respective shapes of functions' values.

Built-in	Value
$\varphi^{\mathsf{ty}}_{c}(c, s)$	p' or $\bot$
$\varphi^{\mathsf{ty}}_{\oplus}(\oplus, \overline{p, s})$	$\{(p'_1, s'_1), \ldots\}$
$\varphi^{\mathsf{ty}}_{\mathsf{encr}}(s)$	$\{p'_1, \ldots\}$
$\varphi^{\mathsf{ev}}_{\oplus}(\oplus, \overline{c, s})$	$\{c'_1, \ldots\}$
$\varphi^{\mathsf{ev}}_{\mathsf{encr}}(c, s)$	$\{c'_1, \ldots\}$
$\varphi^{\mathsf{ev}}_{\mathsf{decr}}(c, s)$	c' or $\bot$

$$\text{(Ev Cxt)}\ \frac{e_1 \underset{\Omega}{\longrightarrow} e_2}{C[e_1] \underset{\Omega}{\longrightarrow} C[e_2]} \qquad \text{(Ev Op)}\ \frac{\varphi^{\mathsf{ev}}_{\oplus}(\oplus, \overline{c, s}) = (c, s)}{\oplus(\overline{c^s}) \underset{\Omega}{\longrightarrow} c^s} \qquad \text{(Ev Enc)}\ \frac{c_2 \in \varphi^{\mathsf{ev}}_{\mathsf{encr}}(c_1, s)}{\mathsf{encr}(c_1^{\varnothing}, s) \underset{\Omega}{\longrightarrow} c_2^s}$$

$$\text{(Ev Decr)}\ \frac{\varphi^{\mathsf{ev}}_{\mathsf{decr}}(c_1, s) = c_2}{\mathsf{decr}(c_1^s) \underset{\Omega}{\longrightarrow} c_2^{\varnothing}} \qquad \text{(Ev OpQuery)}\ \frac{\mathsf{eval}_\theta(\Omega, \theta, \overline{v}) = v}{\theta(\overline{v}) \underset{\Omega}{\longrightarrow} v} \qquad \text{(Ev RecSelect)}\ \{\overline{f : v}\}.f_i \underset{\Omega}{\longrightarrow} v_i$$

$$\text{(Ev Apply)}\ \lambda[d](\overline{x : \kappa}).e(\overline{v}) \underset{\Omega}{\longrightarrow} [\{\overline{v}/\overline{x}\}e]_d \qquad \text{(Ev Tbl)}\ \frac{\Omega(name) = v}{\mathsf{table}(name) \underset{\Omega}{\longrightarrow} v} \qquad \text{(Ev Return)}\ [v]_d \underset{\Omega}{\longrightarrow} v$$

$$\boxed{\begin{aligned} C ::= & [\bullet]_d \mid \oplus(\overline{v}, \bullet, \overline{e}) \mid \theta(\overline{v}, \bullet, \overline{e}) \mid \mathsf{encr}(\bullet, s) \mid \mathsf{decr}(\bullet) \mid \bullet\ (\overline{e}) \\ & \mid v(\overline{v}, \bullet, \overline{e}) \mid \bullet.f \mid \{\overline{f : v}, f : \bullet, \overline{f : e}\} \end{aligned}}$$

Fig. 6. Operational semantics of HYDRA language parameterized by table store Ω: $name \mapsto T\{\overline{f : v}\}$.

Arguments and return values of $\varphi^{\mathsf{ev}}_{\oplus}$ reflect the variety of PHE operations. For instance, `OPE` allows to compare two ciphertexts c_1 and c_2, so we may have $\varphi^{\mathsf{ev}}_{\oplus}(<, c_1, \texttt{OPE}, c_2, \texttt{OPE}) = (\mathsf{true}, \varnothing)$; `ElGamal` allows to multiply a ciphertext c_1 by either another ciphertext c_2 or by a plain text integer, hence, we may have $\varphi^{\mathsf{ev}}_{\oplus}(*, c_1, \texttt{ElGamal}, c_2, \texttt{ElGamal}) = (c_3, \texttt{ElGamal})$ and $\varphi^{\mathsf{ev}}_{\oplus}(*, c_1, \texttt{ElGamal}, 2, \varnothing) = (c_4, \texttt{ElGamal})$; for plaintext operators $s = \varnothing$. Encryption/decryption is modelled by $\varphi^{\mathsf{ev}}_{\mathsf{decr}}(c, s)$ and $\varphi^{\mathsf{ev}}_{\mathsf{encr}}(c, s)$; the former possibly returns a constant and the latter returns a set of constants (for non-deterministic schemes). Note, $\varphi^{\mathsf{ev}}_{\mathsf{decr}}$ and $\varphi^{\mathsf{ev}}_{\mathsf{encr}}$ take $s \in \mathcal{S}_{\varnothing}$, but, naturally $\varphi^{\mathsf{ev}}_{\mathsf{decr}}(c, \varnothing) = \bot$ and $\varphi^{\mathsf{ev}}_{\mathsf{encr}}(c, \varnothing) = \{\}$.

In order to type check HYDRA queries (see §4.2), we introduce a corresponding partial function $\varphi^{\mathsf{ty}}_{\oplus}(\oplus, \overline{p, s})$ that given primitive operator $\oplus$ and its arguments' primitive types and schemes returns the primitive type and scheme of the result (used in (T-OP)), and a $\varphi^{\mathsf{ty}}_{c}(c, s)$ that returns the primitive type of c and, if $s \neq \varnothing$, verifies that c is indeed the ciphertext of s (used in (T-CONST)). As it may be impossible to encrypt values of every primitive type with every s, $\varphi^{\mathsf{ty}}_{\mathsf{encr}}(s)$ specifies the primitive types supported by s. Naturally, for subject reduction, and then to prove security properties and correctness of query transformation we impose three types of correspondence constraints on $\varphi^{\mathsf{ev}}_{\_}$ and $\varphi^{\mathsf{ty}}_{\_}$: *type-preservation* w.r.t. $\varphi^{\mathsf{ty}}_{c}$ and $\varphi^{\mathsf{ty}}_{\oplus}$ for primitive operations $\varphi^{\mathsf{ev}}_{\oplus}$, encryption $\varphi^{\mathsf{ev}}_{\mathsf{encr}}$, and decryption $\varphi^{\mathsf{ev}}_{\mathsf{decr}}$; *progress* of all three $\varphi^{\mathsf{ev}}_{\_}$ when given the right arguments; and *correctness* of encryption/decryption (decryption is the inverse) and PHE operations (correspond to plain-text versions). Notation for built-ins is summarized in Tbl. 3.

3.1.7 Semantics of Relational Operators. Finally, we present our assumptions on the semantics of relational operators. Evaluating projection $\mathsf{eval}_\theta(\Omega, \texttt{proj}, v_t, v_\lambda)$ applies v_λ to every record of v_t. Evaluating $\mathsf{eval}_\theta(\Omega, \texttt{cross}, v_t, v'_t)$ results in a straightforward cross-product of of v_t and v'_t, while $\mathsf{eval}_\theta(\Omega, \texttt{agg}, v_t, f, v_0, v_\lambda)$ takes as input a relation v_t, a field f on which to group, an initial value v_0, and a combining function v_λ which takes a record and an accumulator to produce a new accumulated value. To evaluate filter, $\mathsf{eval}_\theta(\Omega, \texttt{filter}, v_t, v_\lambda)$ returns only those records from relation v_t on which v_λ evaluates to `true`.

3.2 Operational Semantics

Fig. 6 shows the evaluation rules of our language following small-step operational semantics [Plotkin 2004]. Evaluation is parameterized by *table store* Ω mapping table names to corresponding relations from the database. A context C is an expression with a hole $\bullet$; $e = C[e']$ is an expression with the hole occupied by a sub-expression e', i.e., e decomposes into a sub-expression e'

$$\text{(TxSchema)}\ \frac{\overline{s} \in \mathcal{S}}{T\{\overline{f : (p^{\varnothing}, l)}\} \rightsquigarrow_S T\{\overline{f : (p^{s}, l)}\}} \qquad \text{(TxSchemata)}\ \frac{\forall n.\rho(n) \rightsquigarrow_S \rho'(n)}{\rho \rightsquigarrow_S \rho'}$$

$$\mathcal{T} ::= \oplus(\overline{\bullet}) \mid \theta(\overline{\bullet}) \mid \bullet(\overline{\bullet}) \mid \bullet.f \mid \{\overline{f : \bullet}\} \mid T\{\overline{f : \bullet}\}$$

$$\text{(Tx Const)}\ \frac{\varphi^{\mathrm{ev}}(\mathsf{decr}, c_1, s) = c}{c^{\varnothing} \rightsquigarrow c_1^{s}} \qquad \text{(Tx Func)}\ \frac{d \in \mathcal{D} \qquad e \rightsquigarrow e'}{\lambda[\mathsf{CLNT}](\overline{x : \kappa}).e \rightsquigarrow \lambda[d](\overline{x : \kappa'}).e'} \qquad \text{(Tx Encr)}\ \frac{e \rightsquigarrow e' \qquad s \in \mathcal{S}}{e \rightsquigarrow \mathsf{encr}(e', s)}$$

$$\text{(Tx Decr)}\ \frac{e \rightsquigarrow e'}{e \rightsquigarrow \mathsf{decr}(e')} \qquad \text{(Tx Cxt)}\ \frac{\overline{e \rightsquigarrow e'}}{\mathcal{T}[\overline{e}] \rightsquigarrow \mathcal{T}[\overline{e'}]} \qquad \text{(Tx Refl)}\ \frac{e \in \{f, c^{\varnothing}, x, \mathtt{table}(name)\}}{e \rightsquigarrow e} \qquad \text{(Tx Ret)}\ \frac{d \in \mathcal{D} \qquad e \rightsquigarrow e'}{[e]_{\mathsf{CLNT}} \rightsquigarrow [e']_d}$$

Fig. 7. Transformation τ of the subset language to the full language. τ's constraints are defined over types and expressions using relations. Heuristics are instantiations of τ. We assume the original query transforms to a well-typed query which precludes (due to (Tx Encr)) illegal transformations of the sort $\mathsf{encr}(\mathtt{table}(name), s)$.

enclosed in a context C. Evaluation contexts in Fig. 6 are used to define a left-to-right, call-by-value operational semantics. Rule (Ev Cxt) performs a reduction inside a context. Rule (Ev Op) evaluates $\oplus$ based on the evaluation provided by function $\varphi^{\mathrm{ev}}_{\oplus}$. Rule (Ev Enc) encrypts a value with scheme s using $\varphi^{\mathrm{ev}}_{\mathsf{encr}}$ while rule (Ev Decr) decrypts an encrypted value using $\varphi^{\mathrm{ev}}_{\mathsf{decr}}$. Rule (Ev OpQuery) evaluates query operators based on the evaluation provided by eval_{θ} and described in § 3.1.7. Rule (Ev Apply) applies a function to fully evaluated arguments $\overline{v}$ moving the rest of computation inside the appropriate domain. Rule (Ev Return) returns the final result of a computation performed in a possibly different domain to the domain that invoked the computation.

3.3 Query Transformation

The query written by a data analyst (see § 3.1.4) is bereft of security constraints — schemes, domains, and security labels of the database schemata ρ. Before compilation (cf. Fig. 3), Hydra *transforms* the query by filling in security annotations (*blue* parts of Fig. 4). In addition, Hydra chooses the appropriate schemes for input data as part of the transformation.

3.3.1 Transformation Characterization. The transformation consists of a function $\tau[\![\cdot,\cdot]\!]$ taking a query e and plaintext database schemata ρ and returning a related by $\rightsquigarrow$ (see Fig. 7) transformed query e' and related by $\rightsquigarrow_S$ (see Fig. 7) schemata ρ' that uses (encryption) schemes:

Definition 1 (query transformation). *Function* $\tau[\![\cdot,\cdot]\!]$ *is a* query transformation *iff* $\forall \rho, e,$ *and* $(\rho', e') = \tau[\![\rho, e]\!]$, *we have* $e \rightsquigarrow e'$ *and* $\rho \rightsquigarrow_S \rho'$.

On the one hand, Def. 1 gives enough *flexibility* to Hydra's transformation heuristic to account for a variety of external constraints and objectives, such as query execution time and resource availability. On the other hand, by limiting the set of changes using $\rightsquigarrow$ and $\rightsquigarrow_S$ (described shortly), Def. 1 makes it easy to check the preservation of a query's semantics (formallized in § 4.4).

Next, we describe the changes to the query and schemata allowed by $\rightsquigarrow$ and $\rightsquigarrow_S$, respectively. Rules (TxSchemata) and (TxSchema) ensure that the only change to the database schemata is the addition of schemes, in other words, labels of input data columns and primitive types do not change. Rule (Tx Const) allows for encryption of a constant. Rule (Tx Func) allows to change the domain of a lambda expression and also argument types, so the latter match to the inferred labels and encryption schemes. New types $\overline{\kappa'}$ are left unconstrained because the transformed query would still be type-checked. Rules (Tx Encr) and (Tx Decr) allow to introduce encryption or decryption to an arbitrary subexpression within the query. Note that the cases where encryption (or decryption) do not make sense (e.g., `decr(filter(...))` or `encr({...},AES-GCM)`) will be handled by the type

```
agg(filter(cross(table(Customers),
                 filter(table(Orders),
                        λ[SGX](rO: {/* Orders */}). rO.date < 0x..^OPE)),
          λ[SGX](rCO: {/* Customers + Orders */}). rCO.custId == rCO.custKey),
    custId, 0x..^AES-GCM,
    λ[SGX](rP: {price: (Dbl^AES-GCM, High)}, acc: (Dbl^AES-GCM, High)).
          encr(decr(acc) + decr(rP.price), AES-GCM))
```

Fig. 8. Query corresponding to Fig. 5 transformed for CLD execution using explicit security annotations. Types of fields in comments `/* ... */` are taken from the "Security type" column of Tbl. 2.

system (see § 4.2). Rule (Tx Cxt) allows to apply all of the earlier rules to an arbitrary set of subexpressions within the query, while (Tx Refl) handles those that stay intact. It is easy to see that the right hand sides of the above rules for $\rightsquigarrow$ correspond to the result of desugaring of the simplified syntax introduced in §3.1.4. (Tx Ret) allows to change the domain of an ongoing computation; this rule is only used during inductive argument in the proof of transformation correctness (see Th. 3).

3.3.2 Running Example (transformed). Fig. 8 presents a transformation of our running example from Fig. 5; the corresponding transformation of the schemata is presented in "Security type" column of Tbl. 2. Importantly, execution of the transformed query is to be spawned in CLD domain.

We discuss the schemata first. The encryption for High fields is set to AES-GCM, the only allowed in CLD for High (cf. Tbl. 1). For Low fields, the encryption choice is based on the respective usage: custId and custKey participate in equality comparison, hence they are encrypted using a deterministic AES-ECB scheme; date in its turn participates in order comparison, hence an order-preserving OPE is used instead. Apart from types in lambda expressions only adjusted to match the new schemata, there are three key changes in the query: (1) constants representing the date and the initial value to agg are encrypted using (Tx Const); (2) domains of function arguments are replaced with SGX using (Tx Func) since their bodies include non-Public plaintext values; (3) computation inside agg uses decryption and encryption to convert to plaintext and back using (Tx Encr) and (Tx Decr).

As is evident from this example, while our programming language provides core abstractions for streamlined use of different security mechanisms including in combined manner, and — as we show shortly — enables the automated verification of correctness of such use, the level of abstraction is not suited for all. That is, while the full language can be used by system developers with security expertise, it is challenging for many data analysts. We show in the next section that in Hydra all the needed properties hold automatically. In particular, Th. 3 guarantees that the execution of a transformed query (cf. Fig. 8) is equivalent to execution of the underlying short query (Fig. 5), and Th. 1 plus the fact that the full query type-checks guarantee the confidentiality constraints.

4 TYPE SYSTEM AND PROPERTIES

This section presents the properties of program execution enforced by Hydra that reflect end-to-end confidentiality guarantees outlined in §2.1, with the underlying *security type system*.

4.1 Security Framework

Any well-typed program is guaranteed to satisfy confidentiality constraints of inputs w.r.t. a security policy $\mathbb{S}$, namely, no adversary incapable of breaking any of $\mathbb{S}(l)$ accesses l-labelled inputs.

4.1.1 $\mathbb{S}$-Noninterference. As a formal basis, we use an end-to-end property called *noninterference* (NI) [Goguen and Meseguer 1982], its essence being that public outputs are unchanged as secret inputs are varied. By augmenting NI with restrictions imposed by the security policy $\mathbb{S}$, we arrive at a more generic variant, dubbed $\mathbb{S}$-NI, which guarantees that *indistinguishable* outputs are observed by an adversary $A \in \mathcal{A}$ when program executions differ only in inputs at security levels which *A must not be able to access* according to $\mathbb{S}$. Our approach currently does not include a primitive for declassification §2.2.4 and thus intransitive noninterference is not a good design choice for us [Gorrieri and Vernali 2011; Roscoe and Goldsmith 1999]. In Hydra, the inputs are a table store Ω, and the outputs—a final value v of a query e, i.e., $e \xrightarrow[\Omega]{}^* v$. Note, for any binary relation $\sim$ we use $\sim^*$ to denote $\sim$'s reflexive and transitive closure.

4.1.2 Equivalence Relations. Formally we introduce two equivalence relations $\sim^l_\kappa$ for query inputs and, w.r.t. a given $\mathbb{S}$, $\sim^{d,l}$ for query outputs. The two are different because we decouple the following: (a) which input values should remain confidential to certain adversaries (input relation), and (b) which outputs are not observable by certain adversaries (output relation). Note, (a) only depends on security labels, while (b) only depends on domains and encryption schemes. We have $v_1 \sim^l_\kappa v_2$ precisely when v_1 and v_2 differ only in those constants that have confidentiality requirements l. Type κ in $\sim^l_\kappa$ is needed to determine where such constants are, for instance, $\{x : 0\}$ and $\{x : 1\}$ of type κ may or may not be related depending on x's label contained in κ. Only labels of κ are taken into account in $\sim^l_\kappa$. For outputs, $v_1 \sim^{d,l} v_2$ iff v_1 and v_2 inside execution domain d are indistinguishable by any adversary A who must *not* be able to access l according to $\mathbb{S}$ or, equivalently, A cannot distinguish values protected by any of $\mathbb{S}(l)$.

Definition 2. *A non-function value v* satisfies *type κ iff v's structure follows κ's, and for each c^s in v, $\varphi^{\text{ty}}(c, s)$ is equal to the corresponding base type in κ. Table store Ω* satisfies *table environment ρ iff for each name n, $\Omega(n)$ satisfies $\rho(n)$.*

Definition 3 ($\sim^l_\kappa$ and $\sim^{d,l}$ w.r.t. $\mathbb{S}$). *For any v and v' satisfying κ and security policy $\mathbb{S}$, the equivalence relations $v \sim^l_\kappa v'$ and $v \sim^{d,l} v'$* w.r.t. $\mathbb{S}$ *are defined inductively in Fig. 9.*

The key rules are (EquivConst^IN) and (EquivConst^OUT), the remaining rules either say that equal values are equivalent, namely rules (EquivEq^IN), (EquivEq^OUT), (EquivEncEq^IN), and (EquivEncEq^OUT), or propagate the equivalence through the value structure as is the case for rules (EquivTblPW^IN), (EquivTblPW^OUT), (EquivRec^IN), and (EquivRec^OUT). The rule (EquivTblAll^OUT) used in $\sim^{d,l}$ is a little special, we discuss it shortly after. When applied to the two query's inputs, (EquivConst^IN) defines the parts that may differ only allowing variability in constants labelled with l. When applied to the query's outputs, (EquivConst^OUT) restricts which parts of the output may vary, and, notably, to only those protected by domains and encryption schemata deemed, by $\mathbb{S}$, sufficient for l. Rule (EquivTblAll^OUT) establishes equivalence by considering all-to-all correspondence of rows across both tables, this rule is important for `filter`ing using secret-valued predicates and is exactly how our threat model excludes table length information. As an example, for inputs we have $2 \sim^{\texttt{Public}}_{(\texttt{Int},\texttt{Public})} 3$ and $2 \sim^{\texttt{High}}_{(\texttt{Int},\texttt{High})} 3$, but neither $2 \sim^{\texttt{High}}_{(\texttt{Int},\texttt{Public})} 3$ nor $2 \sim^{\texttt{Public}}_{(\texttt{Int},\texttt{High})} 3$. Outputs are more interesting: assuming $\mathbb{S}$ from Tbl. 1, $2 \sim^{\texttt{SGX},\texttt{High}} 3$ and $2 \sim^{\texttt{CLD},\texttt{Public}} 3$ but not $2 \sim^{\texttt{CLD},\texttt{High}} 3$.

We are now in a position to formally state our $\mathbb{S}$-NI property, namely, an expression e satisfies $\mathbb{S}$-NI iff for any two related inputs, i.e., table stores, evaluating e always produces related results. The next definition captures when computation of query e finishing in d satisfies confidentiality requirements of l-labelled inputs for some l in $\mathcal{L}$.

Definition 4 (Level-l $\mathbb{S}$-noninterference $\mathbb{S}$-NI$(e)_{\rho,d,l}$). *Expression e has $\mathbb{S}$-NI$(e)_{\rho,d,l}$ property dubbed* level-l $\mathbb{S}$-noninterference *if and only if for any two stores Ω_1 and Ω_2 satisfying ρ, $\Omega_1 \sim^l_\rho \Omega_2$, and any two values v_1 and v_2, $e \xrightarrow[\Omega_1]{}^* v_1$ and $e \xrightarrow[\Omega_2]{}^* v_2$, it holds that $v_1 \sim^{d,l} v_2$.*

We ultimately want e to satisfy confidentiality requirements of all inputs, Def. 4 for all $l \in \mathcal{L}$.

Definition 5 ($\mathbb{S}$-noninterference $\mathbb{S}$-NI$(e)_{\rho,d}$). *Expression e has* $\mathbb{S}$-noninterference property *$\mathbb{S}$-NI$(e)_{\rho,d}$ if and only if it has level l $\mathbb{S}$-noninterference property $\mathbb{S}$-NI$(e)_{\rho,d,l}$ for every l in $\mathcal{L}$.*

$$\textsc{(EquivConst}^{\textsc{in}}\textsc{)}\ \dfrac{}{c_1^s \sim^{l}_{(p^s,l)} c_2^s} \qquad \textsc{(EquivConst}^{\textsc{out}}\textsc{)}\ \dfrac{(d,s) \in \mathbb{S}(l)}{c_1^s \sim^{d,l} c_2^s} \qquad \textsc{(EquivEq}^{\textsc{in}/\textsc{out}}\textsc{)}\ \dfrac{}{c^\varnothing \sim^{d,l}_{(p^\varnothing,l')} c^\varnothing} \qquad \textsc{(EquivEncEq}^{\textsc{in}/\textsc{out}}\textsc{)}\ \dfrac{\varphi^{\text{ev}}_{\text{decr}}(c_1,s) = \varphi^{\text{ev}}_{\text{decr}}(c_2,s)}{c_1^s \sim^{d,l}_{(p^s,l')} c_2^s}$$

$$\textsc{(EquivRec}^{\textsc{in}/\textsc{out}}\textsc{)}\ \dfrac{\forall i.\ v_i \sim^{d,l}_{\kappa_i} w_i}{\{\overline{f:v}\} \sim^{d,l}_{\{\overline{f:\kappa}\}} \{\overline{f:w}\}} \qquad \textsc{(EquivTblPW}^{\textsc{in}/\textsc{out}}\textsc{)}\ \dfrac{\forall ij.\ v_{ij} \sim^{d,l}_{\kappa_i} w_{ij}}{T\{\overline{f_i : \overline{v_{ij}}^j}^i\} \sim^{d,l}_{T\{\overline{f:\kappa}\}} T\{\overline{f_i : \overline{w_{ij}}^j}^i\}} \qquad \textsc{(EquivTblAll}^{\textsc{out}}\textsc{)}\ \dfrac{\forall ijk.\ v_{ij} \sim^{d,l} w_{ik}}{T\{\overline{f_i : \overline{v_{ij}}^j}^i\} \sim^{d,l} T\{\overline{f_i : \overline{w_{ik}}^k}^i\}}$$

Fig. 9. Equivalence relations $\sim^l_\kappa$ for query input, and $\sim^{d,l}$ w.r.t. $\mathbb{S}$—for output in domain d, from the perspective of an adversary without level l capability. Parts relevant to $\sim^l_\kappa$ ($\sim^{d,l}$) but not to $\sim^{d,l}$ ($\sim^l_\kappa$) are in pink (purple), e.g., (EquivConst$^{\text{OUT}}$) and (EquivTblAll$^{\text{OUT}}$) use only $\sim^{d,l}$. (EquivConst$^{\text{IN}}$) uses only $\sim^l_\kappa$.

4.2 Typing Rules

The security type system for our language must guarantee $\mathbb{S}$-NI property for any well-typed program. While presenting the typing rules we assume a fixed security policy $\mathbb{S}$. Typing judgements are of the form $\rho \wr \Gamma \vdash_d e : \kappa$ where ρ represents relations' schemata, Γ is a typing environment mapping variables to types, d is the domain in which expression e resides, and κ is the type derived for e. We will show in §4.3 that $\rho \vdash_d e : \kappa$ w.r.t. $\mathbb{S}$ implies $\mathbb{S}$-NI$(e)_{\rho,d}$. Fig. 10 presents the typing rules, where we use a convenient shorthand $\kappa \sqsubseteq d$ to assert that $\mathbb{S}$ allows expressions of type κ inside domain d, and we also use the subtyping relation $\kappa <: \kappa'$, which simply propagates $\preccurlyeq$ through type structure. A sequence of typed expressions $\rho \wr \Gamma \vdash_d e_1 : \kappa_1, \rho \wr \Gamma \vdash_d e_2 : \kappa_2, ..., \rho \wr \Gamma \vdash_d e_n : \kappa_n$ is abbreviated as $\rho \wr \Gamma \vdash_d \overline{e : \kappa}$.

Rule (T-TblCall) assigns a type to `table`$(name)$ expression equal to the corresponding schema $\rho(name)$ after checking that the type is allowed in the domain. For corresponding relational values the type is assigned by the (T-Tbl) rule based on types of individual entries. (T-Var) assigns types to variables using the typing environment ensuring that the type is compatible with the domain. (T-Const) assigns values to constants based on built-in type information φ^{ty}_c using the least secure label $\bot$, which is always allowed according to security policy's definition (see §2.2.2). Note, the last two rules do not imply that constants always have $\bot$ label. We present shortly a different rule allowing to bump the confidentiality level; we also rely on it in §4.3 for subject reduction. (T-Fun) types a function based on the body's type, checking also that arguments are allowed in the domain where the function will run. Functions can be typed in any domain d_1 irrespective of the domain d_2 they would execute in. When a (sub)expression is being evaluated in a different domain, as a result of (Ev Apply), (T-Return) changes the typing domain. (T-ConfUp) upgrades to a more confidential type as defined by the subtyping relation; the new type must be allowed in the domain. (T-Apply) mandates that function application returns a result allowed in the domain where the function is invoked. Hence, a function does not carry an explicit label since compatibility checks in the typing rules between security types and domains (attached domain d_2 to the function and context d_1 invoking the function) suffices. (T-Op) types an operator $\oplus$ expression based on $\varphi^{\text{ty}}_\oplus$ and sets the security label to be the lattice join of inputs' security labels. The second premise of (T-Op) says that if *some of* the inputs or the output are encrypted, then all the encryption schemes are the same (equal to s'). (T-Encr) allows encryption of an expression under a scheme s which is compatible with both the security label of the expression and the domain performing encryption. (T-Decr) checks that the type of the plaintext for the encrypted expression is allowed in the domain carrying out the decryption.

$$\boxed{\begin{array}{rl} \kappa \sqsubseteq d & (p^s, l) \sqsubseteq d \Leftrightarrow (d, s) \in \mathbb{S}(l) \\ & T\{\overline{f : (p^s, l)}\} \sqsubseteq d \Leftrightarrow \overline{(p^s, l)} \sqsubseteq d \\ & \{\overline{f : (p^s, l)}\} \sqsubseteq d \Leftrightarrow \overline{(p^s, l)} \sqsubseteq d \\ & \overline{\kappa'} \rightarrow_{d'} \kappa' \sqsubseteq d \end{array}}$$

$$(\textsc{T-Tbl})\ \frac{\forall j.\ \rho \wr \Gamma \vdash_d \overline{v_{i,j} : (p_i^s, l_i)} \qquad \forall i.\ (p_i^s, l_i) \sqsubseteq d}{\rho \wr \Gamma \vdash_d T\{\overline{\overline{f_i : v_{i,j}}^{i}}^{j}\} : T\{\overline{f : (p^s, l)}\}} \qquad (\textsc{T-Var})\ \frac{\Gamma(x) = \kappa \qquad \kappa \sqsubseteq d}{\rho \wr \Gamma \vdash_d x : \kappa}$$

$$(\textsc{T-TblCall})\ \frac{\rho(name) = T\{\overline{f : (p^s, l)}\} \qquad \forall i.\ (p_i^s, l_i) \sqsubseteq d}{\rho \wr \Gamma \vdash_d \mathtt{table}(name) : T\{\overline{f : (p^s, l)}\}} \qquad (\textsc{T-Const})\ \frac{\varphi_c^{\mathrm{ty}}(c, s) = p}{\rho \wr \Gamma \vdash_d c^s : (p^s, \bot)}$$

$$(\textsc{T-Fun})\ \frac{\rho \wr \Gamma, \overline{x : \kappa} \vdash_{d_2} e : \kappa \qquad \forall i.\ \kappa_i \sqsubseteq d_2}{\rho \wr \Gamma \vdash_{d_1} \lambda[d_2](\overline{x : \kappa}).\ e : \overline{\kappa} \rightarrow_{d_2} \kappa} \qquad (\textsc{T-Return})\ \frac{\rho \wr \Gamma \vdash_{d_2} e : \kappa \qquad \kappa \sqsubseteq d_1}{\rho \wr \Gamma \vdash_{d_1} [e]_{d_2} : \kappa}$$

$$(\textsc{T-ConfUp})\ \frac{\rho \wr \Gamma \vdash_d e : \kappa_1 \qquad \kappa_1 <: \kappa_2 \qquad \kappa_2 \sqsubseteq d}{\rho \wr \Gamma \vdash_d e : \kappa_2} \qquad (\textsc{T-Apply})\ \frac{\rho \wr \Gamma \vdash_{d_1} e_\lambda : \overline{\kappa} \rightarrow_{d_2} \kappa \qquad \rho \wr \Gamma \vdash_{d_1} \overline{e : \kappa} \qquad \kappa \sqsubseteq d_1}{\rho \wr \Gamma \vdash_{d_1} e_\lambda(\overline{e}) : \kappa}$$

$$(\textsc{T-Op})\ \frac{\rho \wr \Gamma \vdash_d \overline{e : (p^s, l)} \qquad \varphi_\oplus^{\mathrm{ty}}(\oplus, \overline{p, s}) = (p, s) \qquad s, \overline{s} \in \{s', \varnothing\} \qquad (p^s, \sqcup_i l_i) \sqsubseteq d}{\rho \wr \Gamma \vdash_d \oplus(\overline{e}) : (p^s, \sqcup_i l_i)}$$

$$(\textsc{T-Decr})\ \frac{\rho \wr \Gamma \vdash_d e : (p^s, l) \qquad p \in \varphi_{\mathsf{encr}}^{\mathrm{ty}}(s) \qquad (p, l) \sqsubseteq d}{\rho \wr \Gamma \vdash_d \mathsf{decr}(e) : (p, l)} \qquad (\textsc{T-Encr})\ \frac{\rho \wr \Gamma \vdash_d e : (p^\varnothing, l) \qquad p \in \varphi_{\mathsf{encr}}^{\mathrm{ty}}(s) \qquad (p^s, l) \sqsubseteq d}{\rho \wr \Gamma \vdash_d \mathsf{encr}(e, s) : (p^s, l)}$$

$$(\textsc{T-Filter})\ \frac{\rho \wr \Gamma \vdash_d e_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \qquad \rho \wr \Gamma \vdash_d e_\lambda : \{f_i : (p_i^s, l_i)\}_{i \in I'} \rightarrow_{d'} (\mathsf{Bool}, l) \qquad I' \subseteq I \qquad \forall i.\ (p_i^s, l_i \sqcup l) \sqsubseteq d}{\rho \wr \Gamma \vdash_d \mathtt{filter}(e_t, e_\lambda) : T\{f_i : (p_i^s, l_i \sqcup l)\}_{i \in I}}$$

$$(\textsc{T-Cross})\ \frac{\rho \wr \Gamma \vdash_d e_1 : T\{f_i : (p_i^s, l_i)\}_{i \in I} \qquad \rho \wr \Gamma \vdash_d e_2 : T\{f_j : (p_j^s, l_j)\}_{j \in J} \qquad J \cap I = \emptyset \qquad \forall k \in I \cup J. (p_k^s, l_k \sqcup (\sqcap_{i \in I} l_i) \sqcup (\sqcap_{j \in J} l_j)) \sqsubseteq d}{\rho \wr \Gamma \vdash_d \mathtt{cross}(e_1, e_2) : T\{f_k : (p_k^s, l_k \sqcup (\sqcap_{i \in I} l_i) \sqcup (\sqcap_{j \in J} l_j))\}_{k \in I \cup J}}$$

$$(\textsc{T-Proj})\ \frac{\rho \wr \Gamma \vdash_d e_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \qquad I' \subseteq I \qquad \rho \wr \Gamma \vdash_d e_\lambda : \{f_i : (p_i^s, l_i)\}_{i \in I'} \rightarrow_{d'} \{f_j : (p_j^s, l_j)\}_{j \in J} \qquad \forall j \in J.\ (p_j^s, l_j \sqcup (\sqcap_{i \in I} l_i)) \sqsubseteq d}{\rho \wr \Gamma \vdash_d \mathtt{proj}(e_t, e_\lambda) : T\{f_j : (p_j^s, l_j \sqcup (\sqcap_{i \in I} l_i))\}_{j \in J}}$$

$$(\textsc{T-Agg})\ \frac{\begin{array}{c} \rho \wr \Gamma \vdash_d e_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \qquad \rho \wr \Gamma \vdash_d e_0 : (p^s, l') \qquad I' \cup \{j\} \subseteq I \qquad (p^s, l' \sqcup l_j) \sqsubseteq d \\ \rho \wr \Gamma \vdash_d e_\lambda : (\{f_i : (p_i^s, l_i)\}_{i \in I'}, (p^s, l')) \rightarrow_{d'} (p^s, l') \qquad \varphi_\oplus^{\mathrm{ty}}(=, p_j, s_j, p_j, s_j) = (\mathsf{Bool}, \varnothing) \end{array}}{\rho \wr \Gamma \vdash_d \mathtt{agg}(e_t, f_j, e_0, e_\lambda) : T\{\mathsf{key} : (p_j^{s_j}, l_j), \mathsf{aggVal} : (p^s, l' \sqcup l_j)\}}$$

Fig. 10. Typing judgements for Hydra language excluding two for records. In the top-left, we define relation $\kappa \sqsubseteq d$ meaning expressions of type κ are allowed to appear in domain d by security policy $\mathbb{S}$.

(T-Filter), (T-Cross), (T-Proj), and (T-Agg) type-check query operators `filter`, `cross`, `proj`, and `agg`, respectively. (T-Filter) ensures the test expression e_λ outputs `Bool` and then propagates confidentiality requirements of that output to every field of the resulting relation as the fields' values may depend on e_λ's outcomes. Note, e_λ is allowed to depend on a subset of fields, which can be useful when its domain d' is not the same as `filter`'s d. (T-Cross) is mostly straightforward except for bumping all the labels by $(\sqcap_{i \in I} l_i)$ and $(\sqcap_{j \in J} l_j)$, the meets approximate the security of relation size, we explain them later. (T-Proj) is also straightforward except for another label bumping by $(\sqcap_{i \in I} l_i)$. Finally, (T-Agg) captures the implicit dependency of `aggVal` on `key` by propagating l_j and ensures aggregation on f_j is feasible by requiring values of type $p_j^{s_j}$ to be equality-comparable,

$$\text{(Ext-T-Bracket)}\ \frac{\rho \wr \Gamma \vdash_d e_1 : (p^s, l) \qquad \rho \wr \Gamma \vdash_d e_2 : (p^s, l) \qquad (d_0, s) \notin \mathbb{S}(l)}{\rho \wr \Gamma \Vdash_{d/d_0} \langle e_1 \mid e_2 \rangle : (p^s, l)}$$

$$\text{(Ext-T-Bracket-Enc)}\ \frac{\rho \wr \Gamma \vdash_d e_1 : (p^s, l) \qquad \rho \wr \Gamma \vdash_d e_2 : (p^s, l) \qquad (d_0, \varnothing) \notin \mathbb{S}(l)}{\rho \wr \Gamma \Vdash_{d/d_0} \langle e_1 \mid e_2 \rangle : (p^s, l)}$$

$$\text{(Ext-T-Bracket-Tbl)}\ \frac{\rho \wr \Gamma \vdash_d e_1 : T\{\overline{f : (p^s, l)}\} \qquad \rho \wr \Gamma \vdash_d e_2 : T\{\overline{f : (p^s, l)}\} \qquad (d_0, \varnothing) \notin \mathbb{S}(\sqcap_i l_i)}{\rho \wr \Gamma \Vdash_{d/d_0} \langle e_1 \mid e_2 \rangle : T\{\overline{f : (p^s, l)}\}}$$

Fig. 11. Main typing rules for "bracket" expressions $\langle e_1 \mid e_2 \rangle$ of the extended language.

which usually would mean that if $s \neq \varnothing$, then s is deterministic. Both `filter` and `agg` exhibit *implicit* information flow to the result, correspondingly, from predicate's inputs and values of aggregated column. Such information flows are captured in (T-Filter) and (T-Agg) by appropriately bumping up security labels in the type of the result. The last two primitives replace the typical control-flow constructs based on conditional branching (e.g., "if-then-else" and "switch"), which lets us omit the standard *program-counter* [Hirsch and Cecchetti 2021; Liu et al. 2009; Zheng et al. 2003] approach of tracking the current control branch's label in typing judgments for capturing implicit flows.

It remains to explain the bumping of a security label l of the ouput table's column by $(\sqcap_{i \in I} l_i)$ in rules (T-Cross) and (T-Proj), where $T\{f_i : (p_i^s, l_i)\}_{i \in I}$ is the type of an input table. Bumping is required, because labels capture *not* only the security of the corresponding column's *values*, but also of the *size* of that column, and the latter can be lost without bumping. Even though, we do not protect against leaks through result size, `agg` turns *intermediate* sizes into primitive values, and leaks in primitive values we do protect against. E.g., consider expressions $e_1 = \texttt{proj}(e_t, \lambda(x).\ \{f : \texttt{0}\})$ and $e_2 = \texttt{agg}(e_1, f, \texttt{0}, \lambda(x, y).\ 1 + y)$, type annotations omitted. Without bumping inside `proj`, one would have $e_1 : T\{f : (\texttt{Int}, \bot)\}$ and $e_2 : T\{\texttt{key} : (\texttt{Int}, \bot), \texttt{aggVal} : (\texttt{Int}, \bot)\}$. If e_t's size depended on confidential inputs and e_2 were the final result, the value of `aggVal` would constitute a leak. To protect against such leaks, our type system maintains that the meet of labels of all table's columns is at least as secure as the size of the table; hence, the bumping by $(\sqcap_{i \in I} l_i)$. We did consider having a separate label for the table size, but decided not to further complicate the structure of types.

4.3 Soundness

The following soundness theorem captures end-to-end confidentiality in the execution of computations expressed in Hydra's programming language:

Theorem 1 (Soundness). *If there exists non-function κ, s.t., $\rho \vdash_d e : \kappa$ w.r.t. $\mathbb{S}$ then $\mathbb{S}$-NI$(e)_{\rho,d}$.*

A simple special case is when d alone is sufficient for level-l confidentiality requirements, $(d, \varnothing) \in \mathbb{S}(l)$; we only need l-inputs to be confidential to adversaries who cannot observe final result inside d. Naturally, $\sim^{d,l}$ holds for any values of the same type, and from the subject reduction of Hydra language we get:

Lemma 1 (Inaccessible soundness). *If $(d, \varnothing) \in \mathbb{S}(l)$ and there exists non-function κ, s.t., $\rho \vdash_d e : \kappa$ w.r.t. $\mathbb{S}$ then e has level-l $\mathbb{S}$-noninterference, i.e., $\mathbb{S}$-NI$(e)_{\rho,d,l}$.*

Having degenerate cases handled by Lem. 1, the general structure of the proof for the remaining case, $(d, \varnothing) \notin \mathbb{S}(l)$ is styled after Pottier and Simonet [2002]. First, we extend the language to support two separate branches of execution by adding "bracket" terms $\langle e \mid e \rangle$ and $\langle v \mid v \rangle$ to the syntax of expressions $\mathbb{e}$ and values $\mathbb{v}$, respectively; no nesting allowed. The projections $\lfloor \mathbb{e} \rfloor_i$, $i \in \{1, 2\}$, back to the original language are defined in a straightforward manner.

We extend the type system as shown in Fig. 11 to handle bracket expressions, denoting the new typing judgement with $\Vdash_{d/d_0}$ and abbreviating $\Vdash_{d/d}$ as $\Vdash_d$. Domain d_0 in $\Vdash_{d/d_0}$ denotes the final domain of the computation, where the final value is observed by an attacker, and also where branches were initially typed (cf. Th. 1). As we shortly state in Lem. 4, typeable bracket values in the extended language have indistinguishable (related by $\sim^{d,l}$) branches. (Ext-T-Bracket) ensures that encrypted terms present as branches of a bracket in d would not propagate to the final result in d_0. The last two rules reflect the current $\mathbb{S}$-NI subcase, where d_0 is insufficient for l, $(d, \varnothing) \notin \mathbb{S}(l)$, and an adversary who is able to observe values inside d may still not be allowed to access l. (Ext-T-Bracket-Enc) allows arbitrary encrypted terms present as branches of a bracket as underlying plaintext cannot propagate to d_0. (Ext-T-Bracket-Tbl) is a version of (Ext-T-Bracket-Enc) for branches containing relations of possibly different sizes. In principle, it would be sufficient for (Ext-T-Bracket-Tbl) to have premise $\forall i.(d_0, \varnothing) \notin \mathbb{S}(l_i)$ instead of stricter $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_i l_i)$, if relating well-typed results with (EquivTblAll$^{\text{OUT}}$) was the only goal. The stricter premise is due to subject reduction: (Ev OpQuery) cases involving (Ext-T-Bracket-Tbl)-typed tables rely on $\sqcap_i l_i$ overapproximating security of table sizes (see the discussion on (T-Cross) and (T-Proj) in §4.2).

Then, we define binary encoding taking two non-functional values v_1 and v_2 from the original language and producing a single value $v_1 \star v_2$ from the extended language. If the domain d_0 is observable and initial expressions were typeable *and* only differing in l, then the encoding is also typeable in the extended language in any domain where the corresponding type is allowed assuming the final domain d_0. Formally, we introduce a judgement $\Vdash_{/d_0} v : \kappa$ which holds iff for every d, s.t. $\kappa \sqsubseteq d$, we have $\emptyset \wr \emptyset \Vdash_{d/d_0} v : \kappa$; we abbreviate the latter as simply $\Vdash_{d/d_0} v : \kappa$.

Lemma 2 (Encoding is correct). *If $(d_0, \varnothing) \notin \mathbb{S}(l)$ and for Ω_1 and Ω_2 satisfying ρ we have $\Omega_1 \sim^l_\rho \Omega_2$ then $\Vdash_{/d_0} \Omega_1 \star \Omega_2 : \rho$.*

As the derivation rules for $\Vdash_d$ are a superset of those for $\vdash_d$, we can type the query e itself using the typing rules of the extended language, i.e., $\rho \vdash_d e : \kappa$ implies $\rho \Vdash_d e : \kappa$. There is a small twist, when addressing the inputs labelled with l, we relax $\mathbb{S}$ to $\mathbb{S}^{/l}$ by treating as public all confidentiality levels that do not protect against *all* the adversaries that l protects from.

Definition 6 ($\mathbb{S}^{/l}$). *$\forall l'.\ \mathbb{S}^{/l}(l') = \mathbb{S}(l')$ if $\mathbb{S}(l') \subseteq \mathbb{S}(l)$ and $\mathbb{S}^{/l}(l') = \mathbb{S}(\bot) = \mathcal{D} \times \mathcal{S}_\varnothing$ otherwise.*

Lemma 3. *If $\rho \vdash_d e : \kappa$ w.r.t. $\mathbb{S}$ then $\rho \vdash_d e : \kappa$ w.r.t. $\mathbb{S}^{/l}$.*

The crucial next step is to show that the extended type is preserved by computation.

Theorem 2 (Subject Reduction). *Let $\Vdash_{/d} \mathbb{\Omega} : \rho$, $\rho \wr \Gamma \Vdash_d \mathbb{e} : \kappa$ and $\mathbb{e} \underset{\mathbb{\Omega}}{\Longrightarrow} \mathbb{e}'$ then $\rho \wr \Gamma \Vdash_d \mathbb{e}' : \kappa$.*

The final step is to show that projections of the well-typed values are related.

Lemma 4 (Projections are related). *For non-function $\mathbb{v}$, if there exists κ, s.t. $\rho \Vdash_d \mathbb{v} : \kappa$ w.r.t. $\mathbb{S}^{/l}$, then $\lfloor \mathbb{v} \rfloor_1 \sim^{d,l} \lfloor \mathbb{v} \rfloor_2$.*

Th. 1 then follows from some technical correspondence properties between $\Rightarrow$ and $\longrightarrow$.

4.4 Equivalence of Full Query and Simple Query

For the purpose of Th. 3, we coarsely extend transformation τ to a table store using encrVal which takes an existing table store Ω and transformed table environment ρ' to produce a transformed table store $\Omega' = \mathsf{encrVal}(\Omega, \rho')$.

Theorem 3 (Transformation correctness). *For query transformation $\tau[\![\cdot,\cdot]\!]$, schemata ρ, and expression e, let $(\rho', e') = \tau[\![\rho, e]\!]$. If $\rho' \vdash_d e' : \kappa$ for some domain d and type κ, and also $e \underset{\Omega}{\longrightarrow}^* v$ for some table store Ω satisfying ρ, then for any $\Omega' = \mathsf{encrVal}(\Omega, \rho')$, there exists v', s.t., $e' \underset{\Omega'}{\longrightarrow}^* v'$ and* $\mathsf{decrVal}(v') = v$.

The core steps of the proof of Th. 3 involve showing under some assumptions on encryption and PHE being well-behaved that: (1) $v \rightsquigarrow e'$ implies either e' is a value or can make progress; (2) if $e_1 \rightsquigarrow e'$ and $e_1 \underset{\Omega}{\longrightarrow} e_2$ then e' can make progress and remain a transformation of either e_1 or e_2; (3) evaluation sequence $e'_1 \underset{\Omega'}{\longrightarrow} e'_2 \underset{\Omega'}{\longrightarrow} \ldots$ cannot remain equivalent to the same e indefinitely. (4) $v \rightsquigarrow v'$ implies $\mathsf{decrVal}(v') = v$. A small induction indirection is used to deal with big-step premises used in the definition of $\mathrm{eval}_\theta(\Omega, \theta, \ldots)$.

5 IMPLEMENTATION

We present details of implemented Hydra prototype which extends Apache Spark's SQL library.

5.1 Spark Extension

Spark SQL includes the Dataframe API, and the Catalyst extensible query optimizer which offers a general tree manipulation library. We leverage Spark's query analyzer, optimizer, and execution planner introducing:

- encrypted data sources in the analysis phase (3.2 kLoC Scala),
- logical optimization rules applied during the analysis phase (3.4 kLoC Scala),
- physical optimization rules applied during the planning phase (6.2 kLoC Scala and 100 LoC Java), and
- a code generation step (29 kLoC C++, 2.5 kLoC C, 568 LoC Scala).

Security mechanisms necessary to execute the final Java bytecode are available on the executors: code for PHE computations generated from our custom Catalyst tree expressions is executed in the cloud while appropriate instrumentation is injected via *Java native interface* (JNI) for SGX operations to be invoked in enclaves. Finally, decryption and any necessary post-computation happens on the trusted client side before results are returned.

5.2 Rule-Based Transformation

Following the approach of Catalyst's tree manipulation, Hydra transforms logical plans through *rules* and *strategies*. A rule is a set of "match-replace" transformations applied during analysis phase, while a strategy is a procedure transforming a logical plan to a physical plan. As a first step, the query expressed using the Dataframe API is transformed by Hydra to an internal tree representation called a *logical plan*. Then, Hydra's logical optimization rules are applied to introduce custom security-aware expressions, provide security metadata for every query operator, and encrypt constants. A rule performing static type checking as prescribed by Fig. 10 is then applied to verify the validity of security constraints induced by the combination of security mechanisms. Finally, in the physical planning phase, a strategy is used to map operators to the appropriate security mechanisms, and optionally client-side computation. Each strategy takes into account the output of the heuristic (see § 5.3) and security annotations.

5.3 Query Transformation Heuristics

The transformer (see Fig. 3) uses a heuristic to emit a query with security annotations following our full core programming language. Currently the Hydra prototype implements three heuristics: (1) HydraPHE for "PHE only" mode of execution, using a similar approach to assign PHE schemes to relations' columns as Savvides et al. [2017]; (2) HydraSGX for "SGX only" mode of execution, where all columns are encrypted with AES-GCM analogously to Zheng et al. [2017]; and (3) HydraHYBRID for combining PHE and SGX. HydraHYBRID is guided by empirical measurements of execution times across different security mechanisms. The measurements show that SGX incurs serialization and JNI overheads, and, in PHE, computation is directly on ciphertext while data entering/exiting SGX has to be decrypted/encrypted, favoring PHE sometimes. In general, within the confines of the security policy, HydraHYBRID strives to start a query by using PHE on the untrusted cloud until either hitting a limit of PHE or reaching operations that are faster in SGX. The rest of the query proceeds inside SGX. In the future, we plan to enrich our heuristics by considering e.g., input data size, order of operations in the query, cost of data serialization.

The three implemented heuristics produce a well-typed result only if both CLNT and SGX domains constitute valid escape hatches (see § 2.2.4), i.e. allow plaintext of any security label according to a security policy. While, for increasing technology readiness level, it would be better to have transformation always produce a well-typed query if one exists, we consider it for future work.

5.4 PHE and SGX Operations

We implement PHE schemes in Scala by creating custom Catalyst expressions for homomorphic operations including aggregations, arithmetic calculations, comparisons, conditions, and string matching, thus avoiding using *user-defined functions* (UDFs) which have increased overhead and are opaque to Catalyst optimizations. For SGX operations we implement a shared library in C++ that makes `ecalls` into the enclave while exposing the operations via JNI. The library is then packed into Hydra's *Java archive* (JAR).

5.5 Client-Side Computation

In general, to support client-side computation, a Spark application needs to be broken down into several sub-applications so that the client-side part can be placed in-between. Our current implementation can only produce a single Spark application. Hence, Hydra at the present only supports client-side computation either at the beginning (pre-computation) or at the end (post-computation) of a query. In either case, the relevant data (input columns or intermediate results) is first materialized at the client-side as Scala arrays and decrypted. After pre-computation, the results are appropriately encrypted based on requirements of subsequent computations. Post-computation is performed over plaintext just before the final results are returned to the analyst.

6 EVALUATION

In this section we evaluate the performance of Hydra, addressing three research questions:

RQ1: How does Hydra compare to state-of-the-art systems supporting only a single mechanism?
RQ2: How fast is hybrid execution combining PHE and SGX vs single-mechanism execution?
RQ3: How much effort is saved by Hydra's automated approach compared to explicit programming with security constraints?

6.1 Evaluation Setup

We used an Amazon AWS cluster for evaluation. All reported times in the experiments are averages of 5 executions, and are reported along with error bars.

6.1.1 Comparisons. We use Cuttlefish [Savvides et al. 2017] and Opaque [Zheng et al. 2017] respectively, both built on Apache Spark like Hydra, for comparison.

Cuttlefish introduces *secure data types* (SDTs) which allow programmers to capture properties about the structure and constraints of data, which in turn enable a set of compilation techniques that generate more optimized queries. Specifically, we use Cuttlefish-CS which supports client-side computation/completion for PHE computations for a comparison with HydraPHE. We omit a comparison with Cuttlefish-TH (i.e., Cuttlefish on SGX) since its SGX functionality is very restricted (only re-encryption). Instead, we compare HydraSGX against a more expressive system that supports full-blown relational operators in SGX — Opaque.

Opaque uses SGX for confidentiality and in addition prevents information leakage from access patterns by introducing a set of oblivious operations using *oblivious RAM* (ORAM) [Goldreich 1987]. For fair comparison with Hydra (i.e., HydraSGX), ORAM in Opaque was disabled.

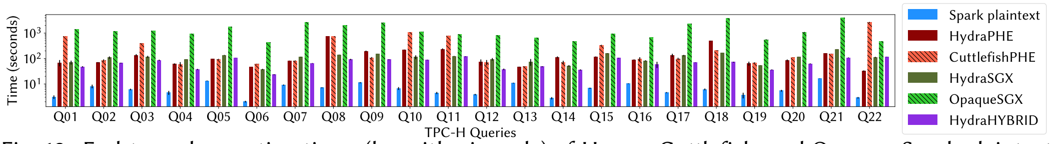

Fig. 12. End-to-end execution times (logarithmic scale) of Hydra, Cuttlefish, and Opaque. Spark plaintext denotes queries run on unencrypted data on vanilla Spark. HydraPHE and CuttlefishPHE use PHE only with client-side completion. HydraSGX and OpaqueSGX use SGX only. HydraHYBRID combines PHE and SGX. All reported times are averages of five executions. Error bars are reported but hard to discern due to the very stable performance and logarithmic scale.

6.1.2 TPC-H Benchmark. We use the popular TPC-H benchmark [TPC 1988] for our evaluation as it is a widely used standard benchmark, also adopted in industry, representative of complex data analytics queries giving answers to critical business questions. In particular, Cuttlefish and Opaque SQL [UC Berkley RISE Lab 2021] have been evaluated using this benchmark, like many other systems (e.g., [Le Quoc et al. 2019; Savvides et al. 2020; Tu et al. 2013]). TPC-H involves 22 queries over 8 tables with a total of 61 columns, holding information of different sensitivities and entropy levels. For fairness of comparison we use the three point lattice of Fig. 2 which corresponds to what is built in throughout Cuttlefish (and immutable), and similarly a policy as in Tbl. 1, ensuring that columns are assigned same schemes (a) as in Cuttlefish for HydraPHE, and (b) as in Opaque for HydraSGX; (c) HydraHYBRID uses a combination of (a) and (b).

6.1.3 Infrastructure. We use an Amazon AWS cluster comprising 10 r5.4xlarge instances as the untrusted cloud for all experiments. Each instance features an Intel Xeon Platinum 8000 series Cascade Lake CPU with 16 vCPUs and 128 GiB of memory, running Ubuntu 18.04 LTS. Along the lines of distributed computations carried out in Zheng et al. [2017], we use Intel SGX-enabled machines in Amazon AWS running Linux SGX SDK version 2.7.101 for all SGX-related computations. For the client-side node we use a single c4.8x large AWS instance featuring an Intel Xeon E5-2666 v3 Haswell CPU with 36 vCPUs and 60 GiB of memory, similarly to the evaluation of Cuttlefish. We use the default AWS network to connect client-side and untrusted cloud via a high speed network connection providing bandwidth up to 10 Gbit/s. To ensure geographical separation, the client-side node (resp. untrusted cloud) was deployed in the availability zone us-east-1b (resp. us-east-1a) of Amazon's North Virginia datacenter. We stored data for evaluation of all systems on AWS S3. We consider the client-side node to be trusted and hence decryption keys are available on this node. The client-side node is used for any client-side computation needed by queries as well as for decrypting the final results before returning them to the data analyst. Decryption keys are passed to the node enclaves via a secure channel after the enclaves are initialized and remotely attested.

6.1.4 Encryption and Attestation. We assume that input data is encrypted once during system setup and made available to the untrusted cloud via AWS S3 and hence we do not include encryption latency in our evaluations. For AES-GCM, AES-ECB, *search on encrypted data* (SWP), and OPE we use 128-bit long keys, and for the asymmetric Paillier and ElGamal 1024-bit long keys. Paillier and ElGamal have a ciphertext expansion factor of 2, leading to 2048-bit ciphertexts [Savvides et al. 2017]. Similarly, SGX enclave attestation and setting up decryption keys happens once at the start of the cloud service and therefore our evaluation does not include these.

6.2 End-To-End Latency (RQ1 and RQ2)

To compare Hydra's performance to state-of-the-art systems and evaluate its different heuristics, we show in Fig. 12 the end-to-end latency of queries on TPC-H (from the time the queries are submitted until the final, decrypted results are returned to the data analyst). Note that the figure includes error bars, which are however hard to discern due to stable performance and a log scale.

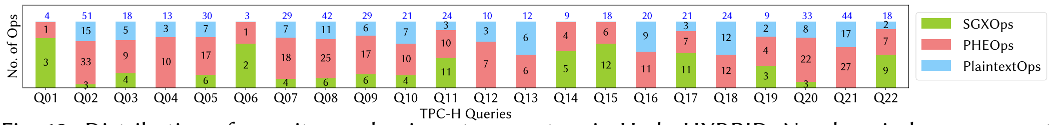

Fig. 13. Distribution of security mechanisms to operators in HydraHYBRID. Numbers in bars represent absolute numbers for respective operator types, dark blue numbers represent totals.

6.2.1 Hydra vs Cuttlefish and Opaque (**RQ1**). Hydra using the "PHE only" heuristic (HydraPHE) is on average 1.6× faster than Cuttlefish. HydraPHE is faster than Cuttlefish on all queries except Q09, Q14, Q17, Q18, and with negligible difference on queries Q04, Q05, Q07, Q08, Q12, and Q21. We attribute Hydra's performance advantage to design choices such as using custom Catalyst tree expressions over UDFs. Cuttlefish's better performance in Q09, Q14, Q17, and Q18 could be due to Spark's built-in optimization outperforming our optimizations. The "SGX only" heuristic (HydraSGX) is on an average 11.3× faster than Opaque. HydraSGX is faster than Opaque on all queries. Hydra's use of Intel SGX SDK [INTEL 2016] instead of Opaque's use of Open Enclave SDK [SDK 2016] along with Hydra using a custom Spark serialization for data going in and out of SGX contribute to HydraSGX's better performance over Opaque.

The primary goal of this comparison is not to show that Hydra is faster than existing systems, but to assert that its generic nature does not introduce an innate penalty over single-mechanism systems, which the results clearly support. Thus we do not dwell on improvements of Hydra's hybrid heuristic (HydraHYBRID) over existing single-mechanism systems despite clear trends (on average 2.7× and 17.9× faster than Cuttlefish and Opaque respectively), but proceed to comparing it to its own single-mechanism heuristics, demonstrating the benefits of combining mechanisms.

6.2.2 Comparison of Heuristics (**RQ2**). Hydra's hybrid execution (HydraHYBRID) is on an average 1.7× and 1.6× faster than HydraPHE and HydraSGX respectively. HydraHYBRID is faster than HydraPHE for all queries except Q22 where it is slower, and Q05 and Q13 where it is performs very closely to HydraPHE. HydraHYBRID is faster than HydraSGX for all queries except Q22 where it performs closely to HydraSGX.

Our evaluation demonstrates that a hybrid approach can not only help overcome limitations in deployment or trust of systems, but also improve performance compared to the use of a single security mechanism.

6.3 Effort (RQ3)

Assessing developer effort is usually far from trivial. To gauge the difficulty of manually identifying and implementing an efficient execution of a query subject to confidentiality constraints using different mechanisms, we show in Fig. 13 a breakdown of security mechanisms (PHE, SGX, none/-plaintext) of operators performed by the respective TPC-H queries in HydraHYBRID. Assignment of security mechanisms to operators in 16 of 22 queries ends up being mixed, while the remaining 6 queries use only PHE (with post-computation). Considering the large number of operators and of possible combinations even with "only" three mechanisms, we believe the analysis conveys how hard it would be for a programmer to choose and correctly implement an efficient combination.

7 RELATED WORK

Many seminal works use (a) client-side computation (cloud for storage only, e.g., [Feldman et al. 2010; Li et al. 2004; Mahajan et al. 2011]), (b) PHE (e.g., [Dong et al. 2016, 2018; Papadimitriou et al. 2016; Popa et al. 2012; Savvides et al. 2020, 2017; Stephen et al. 2014b; Tetali et al. 2013; Tople et al. 2013; Tu et al. 2013]), or (c) trusted hardware (chiefly SGX, e.g., [Arnautov et al. 2016; Baumann et al. 2014; Le Quoc et al. 2019; Lind et al. 2017; Schuster et al. 2015; Shen et al. 2020; Shinde et al. 2017; Silva et al. 2017; Sinha et al. 2015; Tian et al. 2017; Tsai et al. 2017; Zheng et al. 2017]) *individually* for confidentiality-preserving computation. Another complementary line of research uses *differential privacy* (DP) (e.g., [Dwork and Roth 2014; Johnson et al. 2018; Roy et al. 2021]) to *quantitatively* define data privacy and provide probabilistic guarantees, in contrast to our deterministic guarantees (NI). § 6.1.1 (and § 6.2.1) already positioned Hydra with respect to Cuttlefish [Savvides et al. 2017] and Opaque [Zheng et al. 2017], neither of which provides end-to-end formal guarantees.

7.1 Hybrid Approaches

While most works on (b) make combined use of different cryptosystems, and some limited use of (a), to the best of our knowledge, Hydra is first to allow generic combination of mechanisms.

Drucker and Gueron [2017] combine the Paillier scheme with SGX to decouple confidentiality from integrity. The user first encrypts data with Paillier then with a shared key agreed securely with SGX. The latter is unknown to the untrusted application whose sole purpose is to launch the enclave and connect it to the server's OS. Hydra makes more generic use of PHE. Cipherbase [Arasu et al. 2013] provides (an FPGA-based implementation of) trusted hardware that can be used to run a commercial SQL DB system without sacrificing data confidentiality. Given a user-defined security policy, Cipherbase generates a plan to partition query execution between untrusted and trusted machines. This policy allows users to specify data columns to remain in plaintext or encrypted using PHE, in which case computation happens on the untrusted machine. Unlike Hydra, Cipherbase can not deal with data requiring different levels of security or with hypersensitive data unprocessable using PHE or SGX. Orchard [Roth et al. 2020] supports privacy-preserving analytics with DP guarantees against an HbC adversary which is occasionally Byzantine [Lamport et al. 1982]. Orchard transforms queries expressed in Fuzz to use a *collect-and-test* (CaT) primitive [Roth et al. 2019]. Fuzz's linear type system ensures DP guarantees for queries of a certain type. Orchard relies on additive HE (Ring-LWE) [Lyubashevsky et al. 2013] thus limiting query expressivity. Our hybrid model uses a range of PHE schemes while transforming queries for more expressiveness. Unlike Orchard's single untrusted aggregator, Hydra is resilient against many untrusted cloud servers and hence supports distribution beyond only small computations on devices of remote users. Orchard's ratification of DP is sensitive to the underlying functional query language while Hydra's confidentiality is based on strong formal guarantees from NI built upon lambda calculus.

7.2 Languages

Ironclad [Hawblitzel et al. 2014] supports secure applications — written in the high-level Dafny language — with a focus on privacy and integrity via full-system verification to demonstrate *remote equivalence* (RE). RE involves proving both functional correctness and secure information flow. The latter is established using SymDiff [Lahiri et al. 2012] to show NI with declassification for inputs and outputs of the application. While Ironclad strives for stronger guarantees than Hydra including integrity, it requires trusted hardware. Gollamudi and Chong [2016] propose IMP_E, a calculus for expressing programs that leverage SGX-like enclaves. IMP_E includes a type system for enforcing confidentiality in the presence of passive or active attackers by automatically partitioning programs to execute sensitive code in enclaves. IMP_E defines confidentiality restrictions in terms of three fixed security levels L, H and $\top$ similar to the lattice used for TPC-H (cf. Fig. 2). While Hydra focuses more on data processing, it supports custom security labels, and an extensible set of software and hardware mechanisms in a formal system based on lambda and relation abstractions. DFLATE [Gollamudi et al. 2019], a programming model based on flow-limited authorization calculus, enforces strong confidentiality and weak integrity NI guarantees for distributed applications with passive attackers. DFLATE captures guarantees and limitations of underlying *trusted execution environments* (TEEs) in high-level abstractions. Unlike Hydra it does not support software-based cryptosystems. Oak et al. [2021] develop a security-typed language based on Java information flow [Pullicino 2014] to enforce confidentiality and integrity against realistic attackers via robust declassification [Myers et al. 2004] but only for applications specifically using SGX. JSLINQ [Balliu et al. 2016] and SeLINQ [Schoepe et al. 2014] develop formal frameworks based on standard imperative languages to reason about end-to-end confidentiality guarantees of multi-tier web and mobile applications. While drawing inspiration from Pottier and Simonet [2002]'s original work just like our use of NI, both works use fixed security lattices and no security mechanisms. Parker et al. [2019] present LWeb, a Haskell framework (extending the Haskell dynamic information flow control library LIO) for enforcing information flow control policies based on a fixed security lattice for multi-tier web applications. The core of LWeb is formalized in lambda calculus and the proof of NI is shown in Liquid Haskell. Viaduct [Acay et al. 2021] enables programmers to develop programs employing a set of cryptographic mechanisms to ensure confidentiality and integrity with multiple data owners. Viaduct claims but does not formally prove security guarantees of a more general form of NI — non-malleable information flow control [Cecchetti et al. 2017] — a property combining robust declassification and transparent endorsement. Hydra considers only a single data owner and focuses on confidentiality, but supports both software- and hardware-based security mechanisms. Komodo [Ferraiuolo et al. 2017] achieves enclave management in software thus decoupling from hardware requirements. It provides formally verified implementation of software enclaves and uses NI to prove confidentiality and integrity guarantees for them. Komodo does not aim for combining mechanisms. Nickel [Sigurbjarnarson et al. 2018] is a framework for automated verification of intransitive NI, to eliminate covert channels inherent in the OS-application interface. Nickel uses the Z3 SMT solver to prove the NI policy; we use a rigorous manual approach to prove NI formulated from its classical version. For automated verification of NI using Z3, Nickel introduces new proof strategies which increase the *trusted computing base* (TCB). Nickel invokes Z3 to verify noninterference by checking unwinding and refinement conditions. The TCB includes the information flow policy, the checker of unwinding conditions from Nickel, Z3, the checker of refinement conditions from Nickel, and the unverified initialization and glue code. Hydra's TCB is constant and limited to SGX and implementations of PHE schemes.

8 CONCLUSIONS

We presented an approach to using different security mechanisms (e.g., PHE, SGX) to preserve confidentiality in data processing using shared third-party resources. Our approach hinges on a core programming language for confidentiality-preserving computation, and a type system guaranteeing security following a novel generalized form of the theory of NI, namely, $\mathbb{S}$-NI. Data analysts can write queries using a subset of our language without security annotations, transformed without hampering security. We have shown that our approach is competitive with existing systems with hardwired single mechanisms, and can achieve significant speedups over these when combining mechanisms. Our work opens many avenues for future research, several of which we are currently already investigating. Besides the mechanization of $\mathbb{S}$-NI proofs, these include the integration with differential privacy, the inclusion of further properties (e.g., integrity), support for additional mechanisms (e.g., symmetric PHE cryptosystems [Papadimitriou et al. 2016; Savvides et al. 2020]) and for multiple data owners, and the design of broader and more refined heuristics.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and helpful suggestions. The authors would like to thank Marc Arndt for his efforts during the initial stages of the project. This work is supported by Cisco Research University Funding under grant no. 2853380, Hasler Foundation under grant no. 20053, and Meta Security Research under grant (submission) no. 474960397718052.

REFERENCES

Cosku Acay, Rolph Recto, Joshua Gancher, Andrew C. Myers, and Elaine Shi. 2021. Viaduct: an Extensible, Optimizing Compiler for Secure Distributed Programs. In *ACM International Conference on Programming Language Design and Implementation (PLDI '21)*. https://doi.org/10.1145/3453483.3454074

Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security with Cipherbase. In *Conference on Innovative DataSystems Research (CIDR '13)*.

Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *ACM International Conference on Management of Data (SIGMOD '15)*. https://doi.org/10.1145/2723372.2742797

Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

Musard Balliu, Benjamin Liebe, Daniel Schoepe, and Andrei Sabelfeld. 2016. JSLINQ: Building Secure Applications across Tiers. In *ACM Conference on Data and Application Security and Privacy, (CODASPY '16)*.

Andrew Baumann, Marcus Peinado, and Galen C. Hunt. 2014. Shielding Applications from an Untrusted Cloud with Haven. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*.

Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. 2009. Order-Preserving Symmetric Encryption. In *28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '09)*. 224–241.

Ethan Cecchetti, Andrew C. Myers, and Owen Arden. 2017. Nonmalleable Information Flow Control. In *ACM Conference on Computer and Communications Security (CCS '17)*. https://doi.org/10.1145/3133956.3134054

Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag.

Dorothy E. Denning. 1976. A Lattice Model of Secure Information Flow. *Commun. ACM* (1976). https://doi.org/10.1145/360051.360056

Dorothy E. Denning and Peter J. Denning. 1977. Certification of Programs for Secure Information Flow. *Commun. ACM* (1977). https://doi.org/10.1145/359636.359712

Yao Dong, Ana Milanova, and Julian Dolby. 2016. JCrypt: Towards Computation over Encrypted Data. *Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools (PPPJ '16)* (2016). https://doi.org/10.1145/2972206.2972209

Yao Dong, Ana Milanova, and Julian Dolby. 2018. SecureMR: secure mapreduce computation using homomorphic encryption and program partitioning. In *Symposium and Bootcamp on Hot Topics in the Science of Security, HoTSoS 2018*. https://doi.org/10.1145/3190619.3190638

Nir Drucker and Shay Gueron. 2017. Combining Homomorphic Encryption with Trusted Execution Environment: A Demonstration with Paillier Encryption and SGX. In *Workshop on Managing Insider Security Threats (MIST '17)*. https://doi.org/10.1145/3139923.3139933

Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* (2014). https://doi.org/10.1561/0400000042

T. ElGamal. 1985. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *Trans. on Information Theory* (1985). https://doi.org/10.1109/TIT.1985.1057074

Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. 2010. SPORC: Group Collaboration using Untrusted Cloud Resources. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*.

Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. 2017. Komodo: Using Verification to Disentangle Secure-Enclave Hardware from Software. In *Symposium on Operating Systems Principles (SOSP '17)*. https://doi.org/10.1145/3132747.3132782

Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *ACM Symposium on Theory of Computing (STOC '09)*. https://doi.org/10.1145/1536414.1536440

Joseph A. Goguen and José Meseguer. 1982. Security Policies and Security Models. In *IEEE Symposium on Security and Privacy (S&P '82)*. https://doi.org/10.1109/SP.1982.10014

Oded Goldreich. 1987. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *ACM Symposium on Theory of Computing (STOC '87)*. https://doi.org/10.1145/28395.28416

Anitha Gollamudi and Stephen Chong. 2016. Automatic Enforcement of Expressive Security Policies Using Enclaves. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '16)*. https://doi.org/10.1145/2983990.2984002

Anitha Gollamudi, Stephen Chong, and Owen Arden. 2019. Information Flow Control for Distributed Trusted Execution Environments. In *IEEE Computer Security Foundations Symposium (CSF '19)*. https://doi.org/10.1109/CSF.2019.00028

Roberto Gorrieri and Matteo Vernali. 2011. On Intransitive Non-interference in Some Models of Concurrency. In *Foundations of Security Analysis and Design VI - FOSAD Tutorial Lectures*, Alessandro Aldini and Roberto Gorrieri (Eds.). https://doi.org/10.1007/978-3-642-23082-0_5

Chris Hawblitzel, Jon Howell, Jacob R. Lorch, Arjun Narayan, Bryan Parno, Danfeng Zhang, and Brian Zill. 2014. Ironclad Apps: End-to-End Security via Automated Full-System Verification. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*.

Andrew K. Hirsch and Ethan Cecchetti. 2021. Giving semantics to program-counter labels via secure effects. *Proc. ACM Program. Lang.* (2021). https://doi.org/10.1145/3434316

INTEL. 2016. INTEL SGX SDK. Retrieved on 2021-11-10 from https://01.org/intel-software-guard-extensions

Noah M. Johnson, Joseph P. Near, and Dawn Song. 2018. Towards Practical Differential Privacy for SQL Queries. *Int. Conf. on Very Large Data Bases (VLDB)* (2018). https://doi.org/10.1145/3187009.3177733

Shuvendu K. Lahiri, Chris Hawblitzel, Ming Kawaguchi, and Henrique Rebêlo. 2012. SYMDIFF: A Language-Agnostic Semantic Diff Tool for Imperative Programs. In *International Conference on Computer Aided Verification (CAV '12)*. https://doi.org/10.1007/978-3-642-31424-7_54

L. Lamport, R. Shostak, and M. Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* (1982). https://doi.org/10.1145/357172.357176

Do Le Quoc, Franz Gregor, Jatinder Singh, and Christof Fetzer. 2019. SGX-PySpark: Secure Distributed Data Analytics. In *26th International Conference on World Wide Web (WWW '19)*. https://doi.org/10.1145/3308558.3314129

Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. 2004. Secure Untrusted Data Repository (SUNDR). In *Operating Systems Design & Implementation (OSDI'04)*.

Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Christof Fetzer, and Peter R. Pietzuch. 2017. Glamdring: Automatic Application Partitioning for Intel SGX. In *USENIX Annual Technical Conference (ATC '17)*.

Jed Liu, Michael D. George, K. Vikram, Xin Qi, Lucas Waye, and Andrew C. Myers. 2009. Fabric: a platform for secure distributed computation and storage. In *ACM Symposium on Operating Systems Principles , SOSP 2009*.

Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* (2013). https://doi.org/10.1145/2535925

Prince Mahajan, Srinath T. V. Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Michael Dahlin, and Michael Walfish. 2011. Depot: Cloud Storage with Minimal Trust. *ACM Transactions on Computer Systems* (2011). https://doi.org/10.1145/2063509.2063512

Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. 2004. Enforcing Robust Declassification. In *IEEE Computer Security Foundations Workshop (CSFW '04)*. https://doi.org/10.1109/CSFW.2004.9

Aditya Oak, Amir M. Ahmadian, Musard Balliu, and Guido Salvaneschi. 2021. Language Support for Secure Software Development with Enclaves. In *IEEE Computer Security Foundations Symposium (CSF '21)*. https://doi.org/10.1109/CSF51468.2021.00037

Pascal Paillier. 1999. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '99)*. https://doi.org/10.1007/3-540-48910-X_16

Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big Data Analytics over Encrypted Datasets with Seabed. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

James Parker, Niki Vazou, and Michael Hicks. 2019. LWeb: information flow security for multi-tier web applications. *Symp. on Principles of Prog. Lang. (POPL)* (2019). https://doi.org/10.1145/3290388

Gordon D. Plotkin. 2004. A Structural Approach to Operational Semantics. *Journal of Logic and Algebraic Methods Programming* (2004).

Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2012. CryptDB: Processing Queries on an Encrypted Database. *Commun. ACM* (2012). https://doi.org/10.1145/2330667.2330691

François Pottier and Vincent Simonet. 2002. Information Flow Inference for ML. In *Symposium on Principles of Programming Languages (POPL '02)*. https://doi.org/10.1145/503272.503302

Kyle Pullicino. 2014. Jif: Language-based Information-flow Security in Java. *CoRR* abs/1412.8639 (2014). arXiv:1412.8639 http://arxiv.org/abs/1412.8639

A. W. Roscoe and M. H. Goldsmith. 1999. What Is Intransitive Noninterference?. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop, CSFW*. https://doi.org/10.1109/CSFW.1999.779776

Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. 2019. Honeycrisp: Large-scale Differentially Private Aggregation Without a Trusted Core. In *ACM Symposium on Operating Systems Principles (SOSP '19)*. https://doi.org/10.1145/3341301.3359660

Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C. Pierce. 2020. Orchard: Differentially Private Analytics at Scale. In *USENIX Symposium on Operating Systems Design and Implementation, (OSDI '20)*.

Subhajit Roy, Justin Hsu, and Aws Albarghouthi. 2021. Learning Differentially Private Mechanisms. In *IEEE Symposium on Security and Privacy, SP 2021*. https://doi.org/10.1109/SP40001.2021.00060

Ravi S. Sandhu. 1993. Lattice-based Access Control Models. *IEEE Computer* (1993). https://doi.org/10.1109/2.241422

Savvas Savvides, Darshika Khandelwal, and Patrick Eugster. 2020. Efficient Confidentiality-Preserving Data Analytics over Symmetrically Encrypted Datasets. *Proc. VLDB Endow.* (2020). https://doi.org/10.14778/3389133.3389144

Savvas Savvides, Julian James Stephen, Masoud Saeida Ardekani, Vinaitheerthan Sundaram, and Patrick Eugster. 2017. Secure Data Types: A Simple Abstraction for Confidentiality-preserving Data Analytics. In *ACM Symposium on Cloud Computing (SoCC '17)*. https://doi.org/10.1145/3127479.3129256

Daniel Schoepe, Daniel Hedin, and Andrei Sabelfeld. 2014. SeLINQ: Tracking Information across Application-Database Boundaries. In *ACM International Conference on Functional Programming (ICFP '14)*. https://doi.org/10.1145/2628136.2628151

Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud using SGX. In *IEEE Symposium on Security and Privacy (S&P '15)*. https://doi.org/10.1109/SP.2015.10

Open Enclave SDK. 2016. Open Enclave SDK. Retrieved on 2021-11-10 from https://openenclave.io/sdk/

Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. 2020. Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX. In *Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. https://doi.org/10.1145/3373376.3378469

Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. 2017. Panoply: Low-TCB Linux Applications With SGX Enclaves. In *Network and Distributed System Security Symposium, (NDSS '17)*.

Helgi Sigurbjarnarson, Luke Nelson, Bruno Castro-Karney, James Bornholt, Emina Torlak, and Xi Wang. 2018. Nickel: A Framework for Design and Verification of Information Flow Control Systems. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*.

Rodolfo Silva, Pedro Barbosa, and Andrey Brito. 2017. DynSGX: A Privacy Preserving Toolset for Dinamically Loading Functions into Intel(R) SGX Enclaves. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom '17)*. https://doi.org/10.1109/CloudCom.2017.42

Rohit Sinha, Sriram Rajamani, Sanjit Seshia, and Kapil Vaswani. 2015. Moat: Verifying Confidentiality of Enclave Programs. In *ACM Conference on Computer and Communications Security (CCS '15)*. https://doi.org/10.1145/2810103.2813608

Dawn Xiaodong Song, David Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *2000 IEEE Symposium on Security and Privacy (S&P '00)*. 44–55.

Julian James Stephen, Savvas Savvides, Russell Seidel, and Patrick Eugster. 2014a. Practical Confidentiality Preserving Big Data Analysis. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '14)*.

Julian James Stephen, Savvas Savvides, Russell Seidel, and Patrick Th. Eugster. 2014b. Program Analysis for Secure Big Data Processing. In *2014 International Conference on Automated Software Engineering (ASE '14)*. https://doi.org/10.1145/2642937.2643006

Sai Deep Tetali, Mohsen Lesani, Rupak Majumdar, and Todd D. Millstein. 2013. MrCrypt: Static Analysis for Secure Cloud Computations. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '13)*. https://doi.org/10.1145/2509136.2509554

Hongliang Tian, Yong Zhang, Chunxiao Xing, and Shoumeng Yan. 2017. SGXKernel: A Library Operating System Optimized for Intel SGX. In *Conference on Computing Frontiers (CF'17)*. https://doi.org/10.1145/3075564.3075572

Shruti Tople, Shweta Shinde, Zhaofeng Chen, and Prateek Saxena. 2013. AUTOCRYPT: Enabling Homomorphic Computation on Servers to Protect Sensitive Web Content. In *ACM Conference on Computer and Communications Security (CCS'13)*. https://doi.org/10.1145/2508859.2516666

TPC. 1988. TPC-H benchmark. Retrieved on 2021-11-10 from http://www.tpc.org/tpch/

Chia-che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *USENIX Annual Technical Conference (ATC '17)*.

Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing Analytical Queries over Encrypted Data. *Proc. VLDB Endow.* (2013). https://doi.org/10.14778/2535573.2488336

UC Berkley RISE Lab. 2021. MC2. Retrieved on 2022-11-08 from https://mc2-project.github.io/opaque-sql-docs/src/benchmarking/benchmarking.html

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*.

Lantian Zheng, Stephen Chong, Andrew C. Myers, and Steve Zdancewic. 2003. Using Replication and Partitioning to Build Secure Distributed Systems. In *IEEE Symposium on Security and Privacy (S&P 2003)*. https://doi.org/10.1109/SECPRI.2003.1199340

Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*.

Appendix

A Empirical measurements

We designed HydraHYBRID based on an empirical assessment of individual operations' execution times when run using different security mechanisms (see Tbl. 4). For the assessment we used Spark in SGX-enabled single-node setup and a synthetic relation with 1 million rows, encrypted as follows: for plaintext execution data is unencrypted, for PHE it is encrypted under a PHE scheme supporting the specific operation (see "PHE scheme" column), and for SGX we encrypt using AES-GCM.

We observe that PHE is faster than SGX for some operations and slower for others. In particular, for operations using AES-ECB [Daemen and Rijmen 2002], OPE [Boldyreva et al. 2009], and SWP [Song et al. 2000], PHE performs better than SGX. PHE overhead of these operations is relatively low, as the difference with plaintext computation is mostly due to a slightly larger size of operands. In contrast, SGX's overheads due to crossing the enclave boundary are substantial. The situation is the opposite for operations using Paillier and ElGamal: ciphertext size increases substantially and costly multi-precision arithmetic operations are required.

Table 4. Execution times of individual operations using plaintext, PHE, and SGX. "PHE scheme" denotes the scheme data is encrypted under to enable the operation in PHE.

Operation	PHE scheme	Plaintext	PHE	SGX
filter (_ = _)	AES-ECB [Daemen and Rijmen 2002]	5.4 s	5.8 s	8.0 s
filter range	OPE [Boldyreva et al. 2009]	5.7 s	8.7 s	10.7 s
filter match	SWP [Song et al. 2000]	5.4 s	8.8 s	9.4 s
groupby	AES-ECB [Daemen and Rijmen 2002]	6.2 s	7.2 s	57.7 s
sort	OPE [Boldyreva et al. 2009]	7.2 s	13.0 s	41.0 s
select (_ + _)	Paillier [Paillier 1999]	5.2 s	19.4 s	7.8 s
select (_ × _)	ElGamal [ElGamal 1985]	5.1 s	22.4 s	7.8 s

B Complex Lattice

$$\begin{tikzcd}
& \texttt{HighCoETEE} \arrow[dl] \arrow[dr] & \\
\texttt{HighCoE} \arrow[d] & & \texttt{HighTEE} \arrow[ddl] \\
\texttt{LowCoE} \arrow[dr] & & \\
& \texttt{Public} &
\end{tikzcd}$$

$(l \preccurlyeq l')$

Domain $\mathcal{D}$	Scheme $\mathbb{S}^{\emptyset}$	Label $\mathcal{L}$
CLNT	$\emptyset$	HighCoETEE
SGX	Paillier, ElGamal	HighCoETEE
SGX	$\emptyset$	HighTEE
CLD	Paillier, ElGamal	HighCoE
CLD	SWP, OPE, AES-ECB	LowCoE

(a) Complex lattice and its security policy

Fig. 14. A complex lattice and a relation producing a security policy on the right. For brevity we elide some triples of security policy inferrable from $-^{c} \circ \mathbb{S} : \mathcal{L} \longrightarrow \mathcal{A}$ being order- and minimum-preserving.

Fig. 14a gives a slightly more elaborate example that captures a more nuanced requirement of data with $\mathcal{L} = \{\texttt{Public}, \texttt{LowCoE}, \texttt{HighTEE}, \texttt{HighCoE}, \texttt{HighCoETEE}\}$. The suffix "CoE" in the label denotes computation on encrypted data, thus LowCoE refers to deterministic PHE cryptosystems while

HighCoE refers to probabilistic PHE cryptosystems. Computation on data with label HighTEE should be carried out in a TEE (data is in the clear inside TEE), while HighCoETEE means computation should happen on encrypted data within a TEE (data is encrypted inside TEE, as opposed to the case HighTEE). On the one hand, there is definitely a cost to pay when processing encrypted data inside SGX as opposed to the standard model where things are processed plaintext inside SGX. Security requirements, on the other hand, may require such processing (e.g., in [Drucker and Gueron 2017]). This flexibility is made available by our approach. Triples from the second row in Fig. 14a state that data labeled HighCoETEE should be encrypted under Paillier or ElGamal even when residing in SGX. Note, data labeled with the two triples HighCoE can also reside in SGX once encrypted under Paillier or ElGamal.

C Some Omitted Definitions

C.1 Context in expression and encryption/decryption

Fig. 15 presents formal definition of $C[e]$, an expression that results from putting expression e inside context C. Fig. 16 defines $\mathsf{decrVal}(v)$ used in the statement of Th. 3.

$$C[e] ::= \begin{cases} [e]_d & \text{if } C = [\bullet]_d \\ \oplus(\overline{v}, e, \overline{e}) & \text{if } C = \oplus(\overline{v}, \bullet, \overline{e}) \\ \theta(\overline{v}, e, \overline{e}) & \text{if } C = \theta(\overline{v}, \bullet, \overline{e}) \\ \mathsf{encr}(e, s) & \text{if } C = \mathsf{encr}(\bullet, s) \\ \mathsf{decr}(e) & \text{if } C = \mathsf{decr}(\bullet) \\ e(\overline{e}) & \text{if } C = \bullet(\overline{e}) \\ v(\overline{v}, e, \overline{e}) & \text{if } C = v(\overline{v}, \bullet, \overline{e}) \\ e.f & \text{if } C = \bullet.f \\ \{\overline{f : v}, f : e, \overline{f : e}\} & \text{if } C = \{\overline{f : v}, f : \bullet, \overline{f : e}\} \end{cases}$$

Fig. 15. Expression in an evaluation context

C.2 Primitives and Assumptions

C.2.1 Type-preservation assumptions The first set of basic type-preservation assumptions is presented in Fig. 18: (OP-COMP) for primitive operations $\oplus$, (ENCR-COMP)—for encryption encr, and (DECR-COMP)—for decryption decr.

C.2.2 Progress and correctness assumptions The second set of assumptions is in Fig. 19. Rules (OP-PROGRESS) and (ENCR-PROGRESS) ensure that PHE operations and encryption/decryption primitives,

$$\mathsf{decrVal}(v) ::= \begin{cases} c^{\varnothing} & \text{if } v = c^{\varnothing} \\ c_1^{\varnothing} & \text{if } v = c^s \text{ and } \varphi^{\mathrm{ev}}_{\mathsf{decr}}(c, s) = c_1^{\varnothing} \\ \{\overline{f : \mathsf{decrVal}(v)}\} & \text{if } v = \{\overline{f : v}\} \\ T\{\overline{f : \overline{\mathsf{decrVal}(v)}}\} & \text{if } v = T\{\overline{f : \overline{v}}\} \end{cases}$$

Fig. 16. Decryption of the final query's result v.

$$\mathsf{encrVal}(v, \kappa) ::= \begin{cases} c^{\varnothing} & \text{if } v = c^{\varnothing} \text{ and } \kappa = p^{\varnothing} \\ c_1^s & \text{if } v = c^{\varnothing} \text{ and } \kappa = p^s \text{ and } \varphi^{\mathrm{ev}}_{\mathsf{encr}}(c, s) = c_1 \\ \{\overline{f : \mathsf{encrVal}(v, \kappa)}\} & \text{if } v = \{\overline{f : v}\} \text{ and } \kappa = \{\overline{f : \kappa}\} \\ T\{\overline{f : \overline{\mathsf{encrVal}(v, \kappa)}}\} & \text{if } v = T\{\overline{f : \overline{v}}\} \text{ and } \kappa = T\{\overline{f : \kappa}\} \end{cases}$$

Fig. 17. Encryption of the query's inputs, i.e., Ω

$$\textsc{(op-comp)}\ \dfrac{\varphi^{\text{ty}}_{\oplus}(\oplus,\overline{p,s}) = (p,s) \quad \forall i.\ \varphi^{\text{ty}}_{c}(c_i,s_i) = p_i}{\varphi^{\text{ty}}_{c}(\varphi^{\text{ev}}(\oplus,\overline{c,s})) = p} \qquad \textsc{(encr-comp)}\ \dfrac{s \neq \varnothing \quad p \in \varphi^{\text{ty}}_{\mathsf{encr}}(s) \quad \varphi^{\text{ty}}_{c}(c,\varnothing) = p}{\varphi^{\text{ty}}_{c}(\varphi^{\text{ev}}_{\mathsf{encr}}(c,s),s) = p} \qquad \textsc{(decr-comp)}\ \dfrac{s \neq \varnothing \quad p \in \varphi^{\text{ty}}_{\mathsf{encr}}(s) \quad \varphi^{\text{ty}}_{c}(c,s) = p}{\varphi^{\text{ty}}_{c}(\varphi^{\text{ev}}_{\mathsf{decr}}(c,s),\varnothing) = p}$$

Fig. 18. Type preservation for primitive operations.

$$\textsc{(encr-progress)}\ \dfrac{p \in \varphi^{\text{ty}}_{\mathsf{encr}}(s) \quad \varphi^{\text{ty}}_{c}(c,\varnothing) = p}{\varphi^{\text{ev}}_{\mathsf{encr}}(c,s) \neq \bot} \qquad \textsc{(op-progress)}\ \dfrac{\varphi^{\text{ty}}_{\oplus}(\oplus,\overline{p,s}) = (p,s) \quad \forall i.\ \varphi^{\text{ty}}_{c}(c_i,s_i) = p_i}{\varphi^{\text{ev}}_{\oplus}(\oplus,\overline{c,s}) \neq \bot} \qquad \textsc{(eq-correct)}\ \dfrac{\varphi^{\text{ty}}_{\oplus}(=,p,s,p,s) = (\mathsf{Bool},\varnothing) \quad \varphi^{\text{ty}}_{c}(c_1,s) = \varphi^{\text{ty}}_{c}(c_2,s) = p}{\varphi^{\text{ev}}_{\oplus}(=,c_1,s,c_2,s) = (\mathsf{true},\varnothing) \Leftrightarrow c_1 = c_2}$$

$$\textsc{(decr-correct)}\ \dfrac{c' \in \varphi^{\text{ev}}_{\mathsf{encr}}(c,s)}{\varphi^{\text{ev}}_{\mathsf{decr}}(c',s) = c} \qquad \textsc{(phe-correct)}\ \dfrac{\varphi^{\text{ev}}_{\oplus}(\oplus,\overline{c,\varnothing}) = (c,\varnothing) \qquad \varphi^{\text{ev}}_{\oplus}(\oplus,\overline{c',s}) = (c',s) \qquad \forall i.\ c_i^{\varnothing} = \mathsf{decrVal}(c_i'^{\,s_i})}{c^{\varnothing} = \mathsf{decrVal}(c'^{\,s})}$$

Fig. 19. Progress and correctness assumptions for encryption primitives.

$$\textsc{(PrimEv Filter)}\ \dfrac{\forall k \in K. v_\lambda(\{\overline{f_i : v_{i,k}}^{i\in I}\}) \underset{\Omega}{\longrightarrow}^{*} v_k \qquad \forall k \in K. v_k \in \{\mathsf{true},\mathsf{false}\} \qquad K_{\mathsf{true}} = \{k \in K : v_k = \mathsf{true}\}}{\mathsf{eval}_\theta(\Omega, \mathsf{filter}, T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K}\}, v_\lambda) = T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K_{\mathsf{true}}}\}}$$

$$\textsc{(PrimEv Proj)}\ \dfrac{\forall k \in K. v_\lambda(\{\overline{f_i : v_{i,k}}^{i\in I}\}) \underset{\Omega}{\longrightarrow}^{*} \{\overline{f_i : v'_{i,k}}^{i\in J}\}}{\mathsf{eval}_\theta(\Omega, \mathsf{proj}, T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K}\}, v_\lambda) = T\{\overline{\overline{f_i : v'_{i,k}}^{i\in J}}^{k\in K}\}}$$

$$\textsc{(PrimEv Join)}\ \dfrac{I_1 \cap I_2 = \emptyset}{\mathsf{eval}_\theta(\Omega, \mathsf{cross}, T\{\overline{\overline{f_i : v_{i,k}}^{i\in I_1}}^{k\in K_1}\}, T\{\overline{\overline{f_i : v_{i,k}}^{i\in I_2}}^{k\in K_2}\}) = T\{\overline{\overline{f_i : v_{i,k_1}}^{i\in I_1}, \overline{f_i : v_{i,k_2}}^{i\in I_2}}^{k_1,k_2\in K_1\times K_2}\}}$$

$$\textsc{(PrimEv Agg)}\ \dfrac{\{k_{c,1},\dots,k_{c,m_c}\} = \{k : v_{j,k} = c\} \qquad v'_{c,0} = v_0 \qquad v_\lambda(\{\overline{f_i : v_{i,k_{c,s}}}^{i}\}, v'_{c,s-1}) \underset{\Omega}{\longrightarrow}^{*} v'_{c,s}}{\mathsf{eval}_\theta(\Omega, \mathsf{agg}, T\{\overline{\overline{f_i : v_{i,k}}^{i}}^{k\in K}\}, f_j, v_0, v_\lambda) = T\{\overline{\mathsf{key} : c, \mathsf{aggVal} : v'_{c,m_c}}^{c\in\{v_{j,k}:k\in K\}}\}}$$

Fig. 20. Query operator semantics.

respectively, have progress once the arguments' primitive parts are as prescribed by $\varphi^{\text{ty}}_{\mathsf{encr}}$. Rule (eq-correct) imposes special constraints on equality checking represented by symbol =, namely the result is always boolean and reflects the equality of constants. Rules (decr-correct) and (phe-correct) relate the results of such operations to the corresponding plaintexts in a natural way.

C.2.3 Semantics of relational operators Fig. 20 represents the semantics of all the operators. In (PrimEv Filter), a table with $|I|$ columns and $|K|$ rows is filtered to return a table with $|I|$ columns and rows for which the predicate v_λ evaluated to `true`. In (PrimEv Proj), the predicate v_λ outputs a subset of the fields sent as input. In (PrimEv Agg), C is the set of all keys to which rows of the input table map to, v_λ takes in all (one at a time) rows mapped to the same key c and emits the aggregated value for c. Here, s in v_λ ranges over the indices of all rows that mapped to the same key c.

$$\text{(Sub-Base)}\ \frac{l \preccurlyeq l'}{(p^s, l) <: (p^s, l')} \qquad \text{(Sub-Fun)}\ \frac{\kappa <: \kappa' \quad \forall i.\ \kappa_i' <: \kappa_i}{\overline{\kappa} \rightarrow_d \kappa <: \overline{\kappa'} \rightarrow_d \kappa'}$$

$$\text{(Sub-Tbl)}\ \frac{\forall i.\ l_i \preccurlyeq l_i'}{T\{\overline{f : (p^s, l)}\} <: T\{\overline{f : (p^s, l')}\}} \qquad \text{(Sub-Rec)}\ \frac{\forall i.\ l_i \preccurlyeq l_i'}{\{\overline{f : (p^s, l)}\} <: \{\overline{f : (p^s, l')}\}}$$

Fig. 21. Subtyping on security types $\kappa <: \kappa'$.

$$\text{(T-Record)}\ \frac{\rho \wr \Gamma \vdash_d \overline{e : (p^s, l)}}{\Gamma \wr d \vdash_{\overline{\{f:e\}}} \{\overline{f : (p^s, l)}\} :} \qquad \text{(T-RecSelect)}\ \frac{\Gamma \wr d \vdash_{\overline{\{f:e\}}} \{\overline{f : (p^s, l)}\} :}{\Gamma \wr d \vdash_{e.f_i} (p_i^{s_i}, l_i) :}$$

Fig. 22. Record-related typing judgements for Hydra.

C.2.4 Subtyping In Fig. 21 we formally define the subtyping rules w.r.t. partial order on $\mathcal{L}$.

C.2.5 Remaining typing rules In Fig. 22 we present the remaining typing rules. (T-Record) determines a type for a record in essentially the same way as (T-Tbl), and (T-RecSelect) checks the field select.

D Operational Properties of the Core Language

Fig. 18 present correspondence between φ^{ev} and φ^{ty} functions abstracting, respectively, the semantics and types, for built-in constructs (operators, encryption, decryption). Fig. 19 presents assumptions ensuring encryption schemes and PHE operations behave as expected. Fig. 20 presents big-step operational semantics for query operators. The first and the last set of properties are crucial for the subject reduction to go through, while the second is needed to show that query transformation preserves the query's semantics.

Theorem 4 (Subject Reduction). *If Ω satisfies ρ, $\rho \wr \Gamma \vdash_d e : \kappa$ and $e \underset{\Omega}{\longrightarrow} e'$ then $\rho \wr \Gamma \vdash_d e' : \kappa$.*

Proof. Translates straightforwardly from the proof of Th. 8 and Lem. 25 using the following observations: (1) every expression from the original language is a valid expression in the extended language (2) bracket terms can only come from other bracket terms (3) those typing rules of the extended language that do not involve brackets correspond directly to the typing rules of the original language. □

E Transformation correctness

Here our goal is to show

Theorem 5 (Transformation correctness). *For query transformation $\tau[\![\cdot,\cdot]\!]$, schemata ρ, and expression e, let $(\rho', e') = \tau[\![\rho, e]\!]$. If $\rho' \vdash_d e' : \kappa$ for some domain d and type κ, and also $e \underset{\Omega}{\longrightarrow}^* v$ for some table store Ω satisfying ρ, then for any $\Omega' = \mathsf{encrVal}(\Omega, \rho')$, there exists v', s.t., $e' \underset{\Omega'}{\longrightarrow}^* v'$ and* $\mathsf{decrVal}(v') = v$.

Lemma 5 (Well-typedness preserved under context). *For any $e_1, \ldots, e_n$, if $\rho \wr \Gamma \vdash_d \mathcal{T}[\overline{e}] : \kappa$, then there exist $\Gamma_1, \ldots, \Gamma_n$, $\kappa_1, \ldots, \kappa_n$, $d_1, \ldots, d_n$, s.t., for all $e_1', \ldots, e_n'$, $(\forall i. \rho \wr \Gamma_i \vdash_{d_i} e_i' : \kappa_i) \Leftrightarrow \rho \wr \Gamma \vdash_d \mathcal{T}[\overline{e'}] : \kappa$*

Proof. Straightforward case analysis over $\mathcal{T}$. □

Lemma 6 (Transformation context to evaluation context). *For any $\mathcal{T}$, values $v_1', \ldots, v_{k-1}'$, and expressions $e_{k+1}, \ldots, e_n$ there exists C, s.t. for all e $C[e] = \mathcal{T}[v_1', \ldots, v_{k-1}', e, e_{k+1}, \ldots, e_n]$.*

Proof. Straightforward case analysis over $\mathcal{T}$. □

Lemma 7 (Evaluation context to transformation context). *For any C, satisfying $C \notin \{[\bullet]_d, \mathsf{encr}(\bullet, s), \mathsf{decr}(\bullet)\}$, there exists context $\mathcal{T}$, s.t., values $v_1, \dots, v_{k-1}$ and expression $e_{k+1}, \dots, e_n$, s.t. for any e $C[e] = \mathcal{T}[v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n]$*

Proof. Straightforward case analysis over C. □

Lemma 8 (Value preserved under context). *For any $v_i, \dots, v_n$, if $\mathcal{T}[\overline{v}]$ is a non-function value, then for all $v'_1, \dots, v'_n$, $(\forall i.\ v'_i$ is a value $) \Leftrightarrow \mathcal{T}[\overline{v'}]$ is a value.*

Proof. Straightforward case analysis over $\mathcal{T}[\overline{v}]$. □

Lemma 9 (Transformed progress to value). *For any $\rho \rightsquigarrow_S \rho'$, $v \rightsquigarrow e'_1$, and $\Gamma \wr \rho' \vdash_d e'_1 : \kappa$, if v is a value, then either e'_1 is a value or there exists e'_2, s.t., $e'_1 \xrightarrow[\Omega]{} e'_2$ and $v \rightsquigarrow e'_2$.*

Proof. If $v = \lambda[\mathsf{CLNT}](\overline{x : \kappa}).\ e$, then there are two rules matching $\lambda[\mathsf{CLNT}](\overline{x : \kappa}).\ e \rightsquigarrow e'_1$, namely (Tx Cxt) and (Tx Func). In all the cases e'_1 is also a lambda expression, and, hence, a value. In the rest of the proof we assume v is a non-function value.

Induction on $v \rightsquigarrow e'_1$

- Case (Tx Refl), $e_1 = v$, hence, a value.
- Case (Tx Cxt), $v = \mathcal{T}[\overline{v}]$, $e'_1 = \mathcal{T}[\overline{\tilde{e}}]$, and $\overline{v \rightsquigarrow \tilde{e}}$.
  By Lem. 8, each v_i is a value and by Lem. 5, there exist $\Gamma_1, \dots, \Gamma_n$, $\kappa_1, \dots, \kappa_n$, $d_1, \dots, d_n$, s.t., $\rho \wr \Gamma_i \vdash_{d_i} e_i : \kappa_i$ for all i, and we can apply induction hypothesis to $v_i \rightsquigarrow \tilde{e}_i$. There are two cases:
  - Each $\tilde{e}_i$ is a value, hence by Lem. 8 we can conclude that $e'_1 = \mathcal{T}[\overline{\tilde{e}}]$ is a value.
  - There exists some minimal k and $\tilde{e}'$, s.t. $\tilde{e}_k \xrightarrow[\Omega]{} \tilde{e}'$ and $v_k \rightsquigarrow \tilde{e}_k$. Applying Lem. 6 to $\mathcal{T}[\overline{v}]$, $\tilde{e}_1, \dots, \tilde{e}_{k-1}$ and $\tilde{e}_{k+1}, \dots, \tilde{e}_n$ we get that there exists C, s.t.
    $$\mathcal{T}[\overline{\tilde{e}}] = C[\tilde{e}_k] \quad (1a) \qquad C[\tilde{e}'] = \mathcal{T}[\tilde{e}_1, \dots, \tilde{e}_{k-1}, \tilde{e}', \tilde{e}_{k+1}, \dots, \tilde{e}_n] \quad (1b)$$
    We set $e'_2 = C[\tilde{e}']$ and use (Ev Cxt) with (1a) to derive $e'_1 \xrightarrow[\Omega]{} e'_2$ and (Tx Cxt) with (1b) to derive $v \rightsquigarrow e'_2$.
- Case (Tx Const), $e'_1 = c^s$, hence a value, and we are done.
- Case (Tx Encr), $e'_1 = \mathsf{encr}(\tilde{e}'_1, s)$ and $v \rightsquigarrow \tilde{e}'_1$. The only typing rule matching $\mathsf{encr}(\tilde{e}'_1, s)$ is (T-Encr), inverting which we get:
  $$\rho \wr \Gamma \vdash_d \tilde{e}'_1 : (p^{\varnothing}, l) \quad (2a) \qquad \varphi^{\mathrm{ty}}(\mathsf{encr}) = p \longrightarrow p^s \quad (2b)$$
  Applying the induction hypothesis to (2a) we have two cases:
  - Case $\tilde{e}'_1$ is a value.
    The only value shape matching (2a) is $\tilde{e}'_1 = \tilde{c}^{\varnothing}$ and the rule (T-Const), inverting which we get $\varphi^{\mathrm{ty}}(\tilde{c}, \varnothing) = p^{\varnothing}$, combining which with (2b) and (encr-progress) gives $\varphi^{\mathrm{ev}}(\mathsf{encr}, \tilde{c}, s) = c'$ for some c'. The only rule matching $v \rightsquigarrow \tilde{c}^{\varnothing}$ is (Tx Refl), hence $v = \tilde{c}^{\varnothing}$. We set $e'_2 = c'^s$ and use $\varphi^{\mathrm{ev}}(\mathsf{encr}, \tilde{c}, s) = c'$ with (Ev Enc) to derive $e'_1 \xrightarrow[\Omega]{} e'_2$, and with (Tx Const) to derive $v \rightsquigarrow e'_2$.
  - Case $\tilde{e}'_1 \xrightarrow[\Omega]{} \tilde{e}'_2$ and $v \rightsquigarrow \tilde{e}'_2$ for some $\tilde{e}'_2$. We set $e'_2 = \mathsf{encr}(\tilde{e}'_2, s)$, and use (Ev Cxt) to derive $e'_1 \xrightarrow[\Omega]{} e'_2$ and (Tx Encr) to derive $v \rightsquigarrow e_2$.
- Case (Tx Decr), $e'_1 = \mathsf{decr}(\tilde{e}'_1)$ and $v \rightsquigarrow \tilde{e}'_1$. The only typing rule matching $\mathsf{decr}(\tilde{e}'_1)$ is (T-Decr), inverting which we get:
  $$\rho \wr \Gamma \vdash_d \tilde{e}'_1 : (p^s, l) \quad (3a) \qquad \varphi^{\mathrm{ty}}(\mathsf{decr}) = (p^s) \longrightarrow p \quad (3b)$$

Applying the induction hypothesis to (3a) we have two cases:
- Case $\tilde{e}'_1$ is a value.
  The only value shape matching (3a) is $\tilde{e}'_1 = c'^s$.
  The only transformation rule matching $v \rightsquigarrow c'^s$ is (Tx Const), inverting which we get $v = c^\varnothing$ for some c, $\varphi^{\text{ev}}(\mathtt{encr}, c, s) = \tilde{c}$ and, combining with (58) and (3b), also $\varphi^{\text{ev}}(\mathtt{decr}, \tilde{c}, s) = c$. We set $e'_2 = c^\varnothing$ and use (Tx Refl) to derive $v \rightsquigarrow e'_2$ and (Ev Decr) to derive $e'_1 \xrightarrow[\Omega]{} e'_2$.
- Case $\tilde{e}'_1 \xrightarrow[\Omega]{} \tilde{e}'_2$ for some $\tilde{e}'_2$ We set $e'_2 = \mathtt{decr}(\tilde{e}'_2)$, and use (Ev Cxt) to derive $e'_1 \xrightarrow[\Omega]{} e'_2$.

- Case (Tx Func) implies that v is a function value, which has already been handled.

□

Lemma 10 (Transformed terminates). *For any $e \rightsquigarrow e'_1$, $e \rightsquigarrow e'_2$, if $e'_1 \xrightarrow[\Omega]{} e'_2$, then the total number of* encr *and* decr *nodes in e'_2 must be exactly one less than in e'_1.*

Proof. Induction on $e'_1 \xrightarrow[\Omega]{} e'_2$.

- Case (Ev Op), $e'_1 = \oplus(\overline{c^s})$ and $e'_2 = c^s$. The only transformation rule matching $e \rightsquigarrow \oplus(\overline{c^s})$ is (Tx Cxt), hence $e = \oplus(\overline{e})$, but then there is no transformation rule matching $\oplus(\overline{e}) \rightsquigarrow c^s$.
- Case (Ev OpQuery), $e'_1 = \theta(\overline{v})$, $e'_2 = v$, and $\varphi^{\text{ev}}(\theta, \overline{v}) = v$. The only transformation rule matching $e \rightsquigarrow \theta(\overline{c^s})$ is (Tx Cxt), hence $e = \theta(\overline{e})$. The only transformation rule matching $\theta(\overline{e}) \rightsquigarrow v$ is (Tx Cxt), but their right-hand sides are not values, while v is.
- Case (Ev Tbl), $e'_1 = \mathtt{table}(name)$, $e'_2 = v$, and $\Omega(name) = v$. The only transformation rule matching $e \rightsquigarrow \mathtt{table}(name)$ is (Tx Refl), hence $e = \mathtt{table}(name)$. The only transformation rule matching $\mathtt{table}(name) \rightsquigarrow v$ is (Tx Refl), but they right-hand side is not a value, while v is.
- Case (Ev Apply), $e'_1 = \lambda[d](\overline{x : \kappa'}).e'(\overline{v'})$, $e'_2 = [\{\overline{v'}/\overline{x}\}e']_d$. The only transformation rule matching $e \rightsquigarrow \lambda[d](\overline{x : \kappa'}).e'(\overline{v'})$ is (Tx Cxt), all imply that $e = \tilde{e}(\overline{\tilde{e}})$, but there is not transformation rule matching $\tilde{e}(\overline{\tilde{e}}) \rightsquigarrow [\{\overline{v'}/\overline{x}\}e']_d$.
- Case (Ev RecSelect), $e'_1 = \{\overline{f : v}\}.f_i$, $e'_2 = v_i$. The only transformation rule matching $e \rightsquigarrow \{\overline{f : v}\}.f_i$ is (Tx Cxt), hence $e = \tilde{e}.f_i$. The only transformation rule matching $\tilde{e}.f_i \rightsquigarrow v_i$ is (Tx Cxt), but the right-hand side is not a value, while v_i is.
- Case (Ev Enc), $e'_1 = \mathtt{encr}(c^\varnothing, s)$, $e'_2 = c'^s$, the claim follows immediately.
- Case (Ev Decr), $e'_1 = \mathtt{decr}(c^s)$, $e'_2 = \tilde{c}^\varnothing$, the claim follows immediately.
- Case (Ev Cxt), $e'_1 = C[\tilde{e}'_1]$, $e'_2 = C[\tilde{e}'_2]$, and $\tilde{e}'_1 \xrightarrow[\Omega]{} \tilde{e}'_2$.
  Case analysis over $e \rightsquigarrow C[\tilde{e}'_1]$:
  - Cases (Tx Func) and (Tx Const) are impossible.
  - Case (Tx Ret), $e = [\tilde{e}]_{\mathsf{CLNT}}$, $e'_1 = [\tilde{e}'_1]_d$, and $\tilde{e} \rightsquigarrow \tilde{e}'_1$. The only rule matching $\tilde{e} \rightsquigarrow [\tilde{e}'_2]_d$ is (Tx Ret), inverting which we get $\tilde{e} \rightsquigarrow \tilde{e}'_2$. Now we apply an induction hypothesis to $\tilde{e}'_2$ and $\tilde{e}'_1$ to prove that the latter has one less encr and decr node, putting them in the same context C we get the claim.
  - Case (Tx Encr), $e = \tilde{e}$, $e'_1 = \mathtt{encr}(\tilde{e}'_1, s)$, $e'_2 = \mathtt{encr}(\tilde{e}'_2, s)$, and $\tilde{e} \rightsquigarrow \tilde{e}'_1$.
    The only rule matching $\tilde{e} \rightsquigarrow \mathtt{encr}(\tilde{e}'_2, s)$ is (Tx Encr), inverting which we get $\tilde{e} \rightsquigarrow \tilde{e}'_2$. Now we apply an induction hypothesis to $\tilde{e}'_2$ and $\tilde{e}'_1$ to prove that the latter has one less encr and decr node, putting them in the same context C we get the claim.
  - Case (Tx Decr), $e = \tilde{e}$, $e'_1 = \mathtt{decr}(\tilde{e}'_1)$, $e'_2 = \mathtt{decr}(\tilde{e}'_2)$, and $\tilde{e} \rightsquigarrow \tilde{e}'_1$
    The only rule matching $\tilde{e} \rightsquigarrow \mathtt{decr}(\tilde{e}'_2)$ is (Tx Decr), inverting which we get $\tilde{e} \rightsquigarrow \tilde{e}'_2$. Now we apply an induction hypothesis to $\tilde{e}'_2$ and $\tilde{e}'_1$ to prove that the latter has one less encr and decr node, putting them in the same context C we get the claim.

– Case (Tx Cxt), $e = \mathcal{T}[\overline{e}]$, $e'_1 = \mathcal{T}[\overline{e''}]$, and for all i, $e_i \rightsquigarrow e''_i$. It can be concluded that C satisfies the restrictions of Lem. 7, hence we get for $1 \le i \le k-1$ $e_i \rightsquigarrow v''_i$, $e_k \rightsquigarrow \tilde{e}'_1$, and for $k+1 \le i \le n$ $e_i \rightsquigarrow \tilde{e}''_i$.

$$C[\tilde{e}'_2] = \mathcal{T}[v''_1, \dots, v''_{k-1}, \tilde{e}'_2, e''_{k+1}, \dots, e''_n]$$
$$C[\tilde{e}'_1] = \mathcal{T}[v''_1, \dots, v''_{k-1}, \tilde{e}'_1, e''_{k+1}, \dots, e''_n]$$

The only rule matching $\mathcal{T}[\overline{e}] \rightsquigarrow C[\tilde{e'}_2]$ is, naturally, (Tx Cxt), inverting which we get $e_k \rightsquigarrow \tilde{e'}_2$. Now we apply an induction hypothesis to $\tilde{e'}_2$ and $\tilde{e'}_1$ to prove that the latter has one less `encr` and `decr` node, putting them in the same context C we get the claim.

□

Lemma 11 (Transformation context reduction). *If $\mathcal{T}[\overline{e}] \xrightarrow[\Omega]{} e$, then either the root reduction rule is (Ev Cxt) or every e_i is a value.*

Proof. Straightforward case analysis □

Lemma 12 (Transformation and substitution). *If $e \rightsquigarrow e'$ and $\overline{v \rightsquigarrow v'}$, then $\{\overline{v}/\overline{x}\}e \rightsquigarrow \{\overline{v'}/\overline{x}\}e'$.*

Proof. Straightforward case analysis over $e \rightsquigarrow e'$. □

Definition 7 (Query children). Query children *of an evaluation step $e_1 \xrightarrow[\Omega]{} e_2$ are all the derivations of the form $e \xrightarrow[\Omega]{}^* v$ that appear in the premises of (PrimEv Filter), (PrimEv Proj), and (PrimEv Agg) corresponding to the instances of (Ev OpQuery) in $e_1 \xrightarrow[\Omega]{} e_2$.*

Definition 8 (Query height). *Evaluation step $e_1 \xrightarrow[\Omega]{} e_2$ has* query height 0 *iff it has no query children, and* query height $h+1$, $h \ge 0$, *iff all its query children use only evaluation steps with height at most h, and at least one query child has evaluation step with height h.*

Definition 9 (Transformation correctness up to query height). *For any $\rho \rightsquigarrow_S \rho'$, Ω satisfying ρ, $\Omega' = \mathsf{encrVal}(\Omega, \rho')$, $e \rightsquigarrow e'$, and $\Gamma \wr \rho' \vdash_d e' : \kappa$, transformation is* correct up to query height $h \ge 0$ *iff for any $e \xrightarrow[\Omega]{}^* v$ including only evaluation steps with query height* less than h, *there exists v', s.t., $e' \xrightarrow[\Omega']{}^* v'$ and $v \rightsquigarrow v'$.*

Lemma 13 (Transformation correct up to 0). *Transformation is correct up to query height* 0*.*

Proof. As there can be no evaluation steps with query height less than 0, $e = v$. Applying Lem. 9 to $v \rightsquigarrow e'$, we have two cases either e' is a value and we are done, or there exists e'', s.t., $e' \xrightarrow[\Omega']{} e''$ and $v \rightsquigarrow e'$. In the latter case we revert to induction on the total number k of `encr` and `decr` nodes in e'. **Base case,** $k = 0$ is impossible according to Lem. 10. **Inductive case,** $k = k' + 1$ by Lem. 10 e'' has one less `encr` or `decr` node, hence we apply an induction hypothesis to conclude that there exists v', s.t., $v \rightsquigarrow v'$ and $e'' \xrightarrow[\Omega']{}^* v'$, so that also $e' \xrightarrow[\Omega']{}^* v'$. □

Lemma 14 (Encryption is related). *For any Ω satisfying ρ, $\rho \rightsquigarrow_S \rho'$, $\Omega' = \mathsf{encrVal}(\Omega, \rho')$, and any n, $\Omega(n) \rightsquigarrow \Omega'(n)$*

Proof. Induction over $\rho \rightsquigarrow_S \rho'$. □

Lemma 15 (Transformed progress). *For any $h \ge 0$, $\rho \rightsquigarrow_S \rho'$, Ω satisfying ρ, $\Omega' = \mathsf{encrVal}(\Omega, \rho')$, $e_1 \rightsquigarrow e'_1$, and $\Gamma \wr \rho' \vdash_d e'_1 : \kappa$, if transformation is correct up to height h and $e_1 \xrightarrow[\Omega]{} e_2$ has height h, then there exists e'_2, s.t., $e'_1 \xrightarrow[\Omega']{} e'_2$ and either $e_1 \rightsquigarrow e'_2$ or $e_2 \rightsquigarrow e'_2$.*

Proof. Induction over $e_1 \rightsquigarrow e_1'$

- Cases (Tx Const) and (Tx Func) are impossible as values cannot reduce.
- Case (Tx Encr), $e_1' = \mathtt{encr}(\tilde{e'}_1, s)$, $e_1 \rightsquigarrow \tilde{e'}_1$. Applying induction hypothesis to the latter we get $\tilde{e'}_2$, s.t., $\tilde{e'}_1 \xrightarrow[\Omega]{} \tilde{e'}_2$ and either $e_1 \rightsquigarrow \tilde{e'}_2$ or $e_2 \rightsquigarrow \tilde{e'}_2$. We set $e_2' = \mathtt{encr}(\tilde{e'}_2, s)$ and use (Ev Cxt) to derive $\mathtt{encr}(\tilde{e'}_1, s) \xrightarrow[\Omega]{} \mathtt{encr}(\tilde{e'}_2, s)$ and (Tx Encr) to derive either $e_1 \rightsquigarrow \mathtt{encr}(\tilde{e'}_2, s)$ or $e_2 \rightsquigarrow \mathtt{encr}(\tilde{e'}_2, s)$.
- Case (Tx Decr), $e_1' = \mathtt{decr}(\tilde{e'}_1)$, $e_1 \rightsquigarrow \tilde{e'}_1$. Applying induction hypothesis to the latter we get $\tilde{e'}_2$, s.t., $\tilde{e'}_1 \xrightarrow[\Omega]{} \tilde{e'}_2$ and either $e_1 \rightsquigarrow \tilde{e'}_2$ or $e_2 \rightsquigarrow \tilde{e'}_2$. We set $e_2' = \mathtt{decr}(\tilde{e'}_2)$ and use (Ev Cxt) to derive $\mathtt{decr}(\tilde{e'}_1) \xrightarrow[\Omega]{} \mathtt{decr}(\tilde{e'}_2)$ and (Tx Decr) to derive either $e_1 \rightsquigarrow \mathtt{decr}(\tilde{e'}_2)$ or $e_2 \rightsquigarrow \mathtt{decr}(\tilde{e'}_2)$.
- Case (Tx Refl), where subcases $e_1 = f$ or $e_1 = c^{\varnothing}$ or $e_1 = x$ are impossible as they cannot reduce, hence we are left with $e_1 = \mathtt{table}(name)$. The only evaluation rule matching $\mathtt{table}(name) \xrightarrow[\Omega]{} e_2$ is (Ev Tbl), inverting which we get $\Omega(name) = v$. Since $\Omega(name)$ satisfies ρ, by Lem. 14 $\Omega'(name) = v'$ and $v \rightsquigarrow v'$ and we can set $e_2' = v'$.
- Case (Tx Ret), $e_1 = [v]_{\mathrm{CLNT}}$, $e_1' = [\tilde{e}']_d$, and $v \rightsquigarrow \tilde{e}'$. The only typing rule matching $\Gamma \wr \rho' \vdash_d [\tilde{e}']_{:}\kappa$ is (T-Return), inverting which we get $\rho \wr \Gamma \vdash_{\tilde{d}} \tilde{e}' : \kappa$ Now we can apply Lem. 9, and there are two cases:
  - $\tilde{e}'$ is a value. We set $e_2' = \tilde{e}'$, use (Ev Return) to derive $[\tilde{e}']_{\tilde{d}} \xrightarrow[\Omega']{} \tilde{e}'$, and we already have $v \rightsquigarrow \tilde{e}'$.
  - There exists $\hat{e}'$, s.t. $\tilde{e}' \xrightarrow[\Omega']{} \hat{e}'$ and $v \rightsquigarrow \hat{e}'$. We set $e_2' = [\hat{e}']_{\tilde{d}}$, use (Ev Cxt) to derive $e_1' \xrightarrow[\Omega]{} e_2'$ and (Tx Ret) to derive $e_1 \rightsquigarrow e_2'$.
- Case (Tx Cxt), $e_1 = \mathcal{T}[\overline{\tilde{e}}]$, $e_1' = \mathcal{T}[\overline{\tilde{e}'}]$, and $\overline{\tilde{e} \rightsquigarrow \tilde{e}'}$. Applying Lem. 11, there are two cases: either $e_1 \xrightarrow[\Omega]{} e_2$ has (Ev Cxt) as a top reduction rule or every $\tilde{e}_i$ is a value.

  Subcase (Ev Cxt), $e_1 = C[\hat{e}_1]$, $e_2 = C[\hat{e}_2]$, $\hat{e}_1 \xrightarrow[\Omega]{} \hat{e}_2$. By Lem. 7 and injectivity of $\mathcal{T}$, there exist j, s.t., $\tilde{e}_1, \ldots \tilde{e}_{j-1}$ are values and $\tilde{e}_j = \hat{e}_1$. We can apply Lem. 9 to $\tilde{e}_1 \rightsquigarrow \tilde{e}_1', \ldots, \tilde{e}_{j-1} \rightsquigarrow \tilde{e}_{j-1}'$ There are two cases:
  - Each such $\tilde{e}_i'$ is a value. We use Lem. 5 to derive $\Gamma' \wr \rho' \vdash_{d'} \tilde{e}_j' : \kappa_j$ and apply the induction hypothesis to $\tilde{e}_j \xrightarrow[\Omega]{} \hat{e}_2$, getting $\hat{e}'$, s.t., $\tilde{e}_j \xrightarrow[\Omega']{} \hat{e}'$ and either $\hat{e}_1 \rightsquigarrow \hat{e}'$ or $\hat{e}_2 \rightsquigarrow \hat{e}'$. We set $e_2' = C[\hat{e}']$, use (Ev Cxt) to derive $C[\tilde{e}_j] \xrightarrow[\Omega']{} \hat{e}'$, and use (Tx Cxt) to derive either $e_1 \rightsquigarrow e_2'$ or $e_2 \rightsquigarrow e_2'$.
  - There exists some $k < j$ and $\hat{e}'$, s.t., $\tilde{e}_k' \xrightarrow[\Omega']{} \hat{e}'$, $\tilde{e}_k \rightsquigarrow \tilde{e}_k'$, and all $\tilde{e}_1', \ldots \tilde{e}_{k-1}'$ are values. By Lem. 6, there exists C', s.t., $e_1' = C'[\tilde{e}_k']$, $C'[\hat{e}'] = \mathcal{T}[\tilde{e}_1', \ldots \tilde{e}_{k-1}', \hat{e}', \ldots \tilde{e}_n']$. We set $e_2' = C'[\hat{e}']$ and use (Ev Cxt) to derive $e_1' \xrightarrow[\Omega']{} e_2'$, and (Tx Cxt) to derive $e_1 = \mathcal{T}[\overline{\tilde{e}}] \rightsquigarrow e_2'$.

  Subcase, all $\tilde{e}_i$ are values. Let $\tilde{v}_i = \tilde{e}_i$. We can apply Lem. 9 to $\overline{\tilde{e} \rightsquigarrow \tilde{e}'}$. There are two options: either each $\tilde{e}_i'$ is a value, or there exists some j such that $\tilde{e}_j'$ evaluates, we consider the latter first.

  Subsubcase, for some k, $\tilde{e}_k' \rightsquigarrow \hat{e}'$, $\tilde{v}_k \rightsquigarrow \hat{e}'$, and $\tilde{e}_1', \ldots, \tilde{e}_{k-1}'$ are values. This case is exactly the same as the second case in **Subcase (Ev Cxt)**.

  Subsubcase, all $\tilde{e}_i'$ are values. Let $\tilde{v}_i' = \tilde{e}_i'$. Case analysis over $e_1 \xrightarrow[\Omega]{} e_2$.
  - Cases (Ev Tbl), (Ev Enc), (Ev Decr), and (Ev Return) are impossible as there is no matching $\mathcal{T}$.

- Case (Ev RecSelect), $e_1 = \{\overline{f : \tilde{v}}\}.f_i$, $e_2 = \tilde{v}_i$, $e'_1 = \{\overline{f : \tilde{v}'}\}$, and $\overline{\tilde{v} \rightsquigarrow \tilde{v}'}$. The only typing rule matching $\Gamma \wr \rho' \vdash_d \{\overline{f : \tilde{v}'}\} : \kappa$ is (T-Record) inverting which we get $\rho \wr \Gamma \vdash_d \overline{\tilde{v}' : (p^s, l)}$. We set $e'_2 = \tilde{v}'_i$, use (Ev RecSelect) to derive $\{\overline{f : \tilde{v}'}\} \underset{\Omega'}{\longrightarrow} \tilde{v}'_i$, and we already have $\tilde{v}_i \rightsquigarrow \tilde{v}'_i$.
- Case (Ev Op), $e_1 = \oplus(\overline{c^s})$, $e_2 = c^s$, $e'_1 = \oplus(\overline{\tilde{v}'})$, $\varphi^{\text{ev}}(\oplus, \overline{c^s}) = c^s$, and $\overline{c^s \rightsquigarrow ve'}$. The only typing rule matching $\Gamma \wr \rho' \vdash_d \oplus(\overline{\tilde{v}'}) : \kappa$ is (T-Op), inverting which we get for some $\tilde{s}'$:
$$\rho \wr \Gamma \vdash_d \overline{\tilde{v}' : (p^{\tilde{s}}, l)} \qquad (5a) \qquad\qquad \varphi^{\text{ty}}(\oplus) = \overline{p^{\tilde{s}}} \rightarrow p^{\tilde{s}} \qquad (5b)$$
The only typing rule matching (5a) is (T-Const), inverting which we get $\overline{\tilde{v}' = c'_{\tilde{s}}}$ and $\varphi^{\text{ty}}(c, \tilde{s}) = p^{\tilde{s}}$. Combining the latter with (5b) and using (op-progress), we get there exists some $\hat{c}$ and $\hat{s}$, s.t., $\varphi^{\text{ev}}(\oplus, \overline{c, s}) = \hat{c}_{\hat{s}}$. We set $e'_2 = \hat{c}_{\hat{s}}$, using (Ev Op) to derive $\oplus(\overline{c'_{\tilde{s}}}) \underset{\Omega'}{\longrightarrow} e'_2$. The only transformation rules matching $\overline{c^s \rightsquigarrow c'_{\tilde{s}}}$ are (Tx Const) and (Tx Refl), both imply that $\overline{s = \varnothing}$ and $\overline{c_{\varnothing} = \text{decrVal}(c'_{\tilde{s}})}$. Hence, we can apply (phe-correct) to the latter deriving $c_s = \text{decrVal}(\hat{c}_{\hat{s}})$, finally, the latter allows us to use (Tx Const) for deriving $c_s \rightsquigarrow \hat{c}_{\hat{s}}$.
- Case (Ev OpQuery) $e_1 = \theta(\overline{v})$, $e'_1 = \theta(\overline{\tilde{v}'})$, $\varphi^{\text{ev}}(\theta, \overline{v}) = v$, and $e_2 = v$. Case analysis over θ:
  * Case $\theta = \texttt{filter}$, by inversion of (PrimEv Filter)
$$\forall k.\ v_2(\{\overline{f_j : v_{j,k}}^{j\in J}\}) \underset{\Omega}{\longrightarrow}^{*} v^*_k \qquad (6a) \qquad\qquad K_{\texttt{true}} = \{k \in K : v^*_k = \texttt{true}\} \qquad (6b)$$
$$v_1 = T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K}\} \qquad (6c) \qquad\qquad \forall k.\ v_k \in \{\texttt{true}, \texttt{false}\} \qquad (6d)$$
$$e_2 = T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K_{\texttt{true}}}\} \qquad (6e)$$
The only rule, matching $\tilde{v}_1 \rightsquigarrow T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K}\}$ is (Tx Cxt), inverting which we get $\overline{\overline{v \rightsquigarrow \tilde{v}}}$, hence we can use (Tx Cxt) twice to derive for each k, $v_2(\{\overline{f_j : v_{j,k}}^{j\in J}\}) \rightsquigarrow \tilde{v}_2(\{\overline{f_j : \tilde{v}_{j,k}}^{j\in J}\})$. Note, that since $e_1 \underset{\Omega}{\longrightarrow} e_2$ is has height h, (6a) has height at most $(h-1)$, so we can use an assumption that transformation is correct up to the level h to derive that for each k, there exists $\tilde{v}^*_k$, s.t. $\tilde{v}_2(\{\overline{f_j : \tilde{v}_{j,k}}^{j\in J}\}) \underset{\Omega'}{\longrightarrow} \tilde{v}^*_k$ and $v^*_k \rightsquigarrow \tilde{v}^*_k$. The only transformation rule matching $\overline{v^* \rightsquigarrow \tilde{v}^*}$ and Equation 6d is (Tx Refl), hence $\overline{\tilde{v}^* = v^*}$. We set $e'_2 = T\{\overline{\overline{f_i : \tilde{v}_{i,k}}^{i\in I}}^{k\in K_{\texttt{true}}}\}$ use (PrimEv Filter) and (Ev OpQuery) to derive $e'_1 \underset{\Omega}{\longrightarrow} e'_2$ and (Tx Cxt) to derive $e_2 \rightsquigarrow e'_2$.
  * Case $\theta = \text{agg}$, by inversion of (PrimEv Agg):
$$v_1 = T\{\overline{\overline{f_i : v_{i,k}}^{i}}^{k\in K}\} \qquad (7a) \qquad\qquad v_2 = f_j \qquad (7b) \qquad\qquad v^*_{c,0} = v_3 \qquad (7c)$$
$$v_4(\{\overline{f_i : v_{i,k_{c,s}}}^{i}\}, v^*_{c,s-1}) \underset{\Omega}{\longrightarrow}^{*} v^*_{c,s} \qquad (7d) \qquad\qquad \{k_{c,1}, \ldots, k_{c,m_c}\} = \{k : v_{j,k} = c\} \qquad (7e)$$
The only transformation rule matching $T\{\overline{\overline{f_i : v_{i,k}}^{i}}^{k\in K}\} \rightsquigarrow \tilde{v}_1$ is (Tx Cxt), inverting which we get $\tilde{v}_1 = T\{\overline{\overline{f_i : \tilde{v}_{i,k}}^{i}}^{k\in K}\}$ and $\overline{\overline{v \rightsquigarrow \tilde{v}}}$.
It is straightforward to show for each c by induction on s that there exist $\tilde{v}^*_{c,s}$, s.t., $v^*_{c,s} \rightsquigarrow \tilde{v}^*_{c,s}$ and $\tilde{v}_4(\{\overline{f_i : \tilde{v}_{i,k_{c,s}}}^{i}\}, \tilde{v}^*_{c,s-1}) \underset{\Omega}{\longrightarrow}^{*} \tilde{v}^*_{c,s}$. **Base case** follows from $v_3 \rightsquigarrow \tilde{v}_3$ **Inductive case**

use the fact that each of (7d) has height at least h and transformation is assumed to be correct up to height h. The set C of keys is the same in the transformed case due to (EQ-CORRECT). We set $e'_2 = T\{\overline{\texttt{key} : c, \texttt{aggVal} : \tilde{v}^*_{c,m_c}}^{c\in C}\}$ and use (PrimEv Agg) with (Ev OpQuery) to show $e'_1 \rightsquigarrow e'_2$, we finally use (Tx Cxt) to show $e_2 \rightsquigarrow e'_2$.

$*$ Case $\theta = \texttt{cross}$, by inversion of (PrimEv Join) $I_1 \cap I_2 = \emptyset$, and:

$$v_1 = T\{\overline{\overline{f_i : v_{i,k}}^{i\in I_1}}^{k\in K_1}\} \quad (8a) \qquad v_2 = T\{\overline{\overline{f_i : v_{i,k}}^{i\in I_2}}^{k\in K_2}\} \quad (8b)$$

$$e_2 = T\{\overline{\overline{f_i : v_{i,k_1}}^{i\in I_1}, \overline{f_i : v_{i,k_2}}^{i\in I_2}}^{k_1,k_2\in K_1\times K_2}\} \quad (8c)$$

The only rule matching $T\{\overline{\overline{f_i : v_{i,k}}^{i\in I_r}}^{k\in K_r}\} \rightsquigarrow \tilde{v}_r$ is (Tx Cxt), hence $v_r = T\{\overline{\overline{f_i : \tilde{v}_{i,k}}^{i\in I_r}}^{k\in K_r}\}$, and $\overline{\overline{v_{i,k} \rightsquigarrow \tilde{v}_{i,k}}^{i\in I_r}}^{k\in K_r}$. We set $e'_2 = T\{\overline{\overline{f_i : \tilde{v}_{i,k_1}}^{i\in I_1}, \overline{f_i : \tilde{v}_{i,k_2}}^{i\in I_2}}^{k_1,k_2\in K_1\times K_2}\}$ and use (PrimEv Join) and (Ev OpQuery) to derive $e'_1 \xrightarrow[\Omega']{} e'_2$ and (Tx Cxt) to derive $e_2 \rightsquigarrow e'_2$.

$*$ Case $\theta = \texttt{proj}$, by inversion of (PrimEv Proj):

$$v_1 = T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K}\} \quad (9a) \qquad v_2(\{\overline{f_i : v_{i,k}}^{i\in I}\}) \xrightarrow[\Omega]{}{}^{*} \{\overline{f_i : v'_{i,k}}^{i\in J}\} \quad (9b)$$

$$e_2 = T\{\overline{\overline{f_i : v'_{i,k}}^{i\in J}}^{k\in K}\} \quad (9c)$$

The only rule, matching $\tilde{v}_1 \rightsquigarrow T\{\overline{\overline{f_i : v_{i,k}}^{i\in I}}^{k\in K}\}$ is (Tx Cxt), inverting which we get $\overline{\overline{v \rightsquigarrow \tilde{v}}}$, hence we can use (Tx Cxt) twice to derive for each k, $v_2(\{\overline{f_i : v_{i,k}}^{i\in I}\}) \rightsquigarrow \tilde{v}_2(\{\overline{f_i : \tilde{v}_{i,k}}^{i\in I}\})$. Note, that since $e_1 \xrightarrow[\Omega]{} e_2$ is has height h, (9a) has height at most $(h-1)$, so we can use an assumption that transformation is correct up to the level h to derive that for each k, there exists $\tilde{v}^*_k$, s.t. $\tilde{v}_2(\{\overline{f_i : \tilde{v}_{i,k}}^{i\in I}\}) \xrightarrow[\Omega']{} \tilde{v}^*_k$ and $\{\overline{f_i : v'_{i,k}}^{i\in J}\} \rightsquigarrow \tilde{v}^*_k$. The only transformation rule matching $\{\overline{f_i : v'_{i,k}}^{i\in J}\} \rightsquigarrow \tilde{v}^*_k$ is (Tx Cxt), hence $\overline{\tilde{v}^* = \{\overline{f_i : \tilde{v}'_{i,k}}^{i\in J}\}}$ for some $\tilde{v}'_{i,k}$. We set $e'_2 = T\{\overline{\overline{f_i : \tilde{v}'_{i,k}}^{i\in J}}^{k\in K}\}$ and use (PrimEv Proj) and (Ev OpQuery) to derive $e'_1 \xrightarrow[\Omega']{} e'_2$ and (Tx Cxt) to derive $e_2 \rightsquigarrow e_2$.

– Case (Ev Apply), $e_1 = \lambda[d](\overline{x : \kappa}).e(\overline{v})$, $e'_1 = \tilde{v}'_\lambda(\overline{\tilde{v}'})$, $\lambda[d](\overline{x : \kappa}).e \rightsquigarrow \tilde{v}'$, $\overline{e \rightsquigarrow \tilde{v}'}$, and $e_2 = [\{\overline{v}/\overline{x}\}e]_d$. The only typing rule matching the shape of e'_1 is (T-Apply), inverting which we get

$$\rho \wr \Gamma \vdash_d \tilde{v}'_\lambda : \overline{\tilde{\kappa}} \to_{\tilde{d}} \tilde{\kappa} \quad (10a) \qquad \rho \wr \Gamma \vdash_d \overline{\tilde{v}' : \kappa} \quad (10b)$$

The only typing rule matching (10a) is (T-Fun), inverting which we get $\tilde{v}'_\lambda = \lambda[\tilde{d}](\overline{x : \tilde{\kappa}}).\, \tilde{e}'$. We set $e'_2 = [\{\overline{\tilde{v}'}/\overline{x}\}\tilde{e}]_{\tilde{d}}$ and use (Ev Apply) to derive $e'_1 \xrightarrow[\Omega']{} e'_2$. The only transformation rule matching $\lambda[d](\overline{x : \kappa}).\, e \rightsquigarrow \lambda[\tilde{d}](\overline{x : \tilde{\kappa}}).\, \tilde{e}'$ is (Tx Func), inverting which we get $e \rightsquigarrow \tilde{e}'$ and $d = \texttt{CLNT}$. We apply Lem. 12 to derive $\{\overline{v}/\overline{x}\}e \rightsquigarrow \{\overline{\tilde{v}'}/\overline{x}\}\tilde{e}$, and then use (Tx Ret), to derive $e_2 \rightsquigarrow e'_2$.

□

Lemma 16. *For any two values v and v', if $v \rightsquigarrow v'$, then* $\text{decrVal}(v') = v$.

$$\begin{array}{rcl} \text{Value} & \mathbb{v} & ::= & T\{\overline{\overline{f : \mathbb{v}}}\} \mid \{\overline{f : \mathbb{v}}\} \mid \lambda[d](\overline{x : \kappa}).\, \mathbb{e} \mid \\ & & & \langle v \mid v \rangle \mid v \\ \text{Expression} & \mathbb{e} & ::= & \mathbb{v} \mid \mathbb{e}(\overline{\mathbb{e}}) \mid \oplus(\overline{\mathbb{e}}) \mid \{\overline{f : \mathbb{e}}\} \mid \mathbb{e}.f \mid \theta(\overline{\mathbb{e}}) \mid \\ & & & \mathtt{encr}(\mathbb{e}, s) \mid \mathtt{decr}(\mathbb{e}) \mid [\mathbb{e}]_d \mid \langle e \mid e \rangle \mid e \end{array}$$

Fig. 23. Expressions and values in the extended language.

Proof. Straightforward induction on $v \rightsquigarrow v'$. □

Proof. Proof of Th. 3.

We first prove that transformation is correct up to an arbitrary query height h using induction on h. **Base case** follows from Lem. 13. **Inductive case** we assume that transformation is correct up to a level h and we show that it is correct up to a level $h + 1$. Consider some $e \underset{\Omega}{\longrightarrow}^* v$ that only includes evaluation steps with query height at most h and $e \rightsquigarrow e'$, next we perform an induction over $e \underset{\Omega}{\longrightarrow}^* v$, where the base case follows from the same argument as in the proof of *Lem. 13*. Hence, $e \underset{\Omega}{\longrightarrow} \tilde{e} \underset{\Omega}{\longrightarrow}^* v$. We apply Lem. 15 to $e \underset{\Omega}{\longrightarrow} \tilde{e}$ and $e \rightsquigarrow e'$ and get some $\tilde{e}'$, s.t., $e' \underset{\Omega'}{\longrightarrow} \tilde{e}'$, more over, due to Th. 4 $\Gamma \wr \rho' \vdash_d \tilde{e}' : \kappa$. There are two cases:

- Case $\tilde{e} \rightsquigarrow \tilde{e}'$. We apply the induction hypothesis to $\tilde{e} \underset{\Omega}{\longrightarrow}^* v$ and we are done.
- Case $e \rightsquigarrow \tilde{e}'$. We perform inner induction on the number of decr and encr nodes $\tilde{e}$ has. By Lem. 10, there is one less such node each time $e \rightsquigarrow \tilde{e}'$, hence, after a finite number of steps we will get $\tilde{e} \rightsquigarrow \tilde{e}'$, when we would proceed as in the first case.

We have, thus, shown that the transformation is correct up to an arbitrary query height h.

Now consider an arbitrary ρ, a table store Ω satisfying ρ, and expression e. Let $(\rho', e') = \tau[\![\rho, e]\!]$ and $\Omega' = \mathrm{encrVal}(\Omega, \rho')$. By the definition of a query transformation, we know that $e \rightsquigarrow e'$ and $\rho \rightsquigarrow_S \rho'$. Using the assumption that $e \underset{\Omega}{\longrightarrow}^* v$ and $\rho' \vdash_d e' : \kappa$ and the fact that transformation is correct up to an arbitrary query height we conclude that there exists v', s.t., $e' \underset{\Omega'}{\longrightarrow}^* v'$ and $v \rightsquigarrow v'$. It remains to apply Lem. 16 to get $\mathrm{decrVal}(v') = v$. □

F Soundness Proof

F.1 Extended language

Extension to our core language (see Fig. 4) is presented in Fig. 23, in essence, we add a bracket construct representing two evaluating programs to the set of values and expressions.

There is a projection function presented in Fig. 24 that allows us to recover either branch of the extended language, and an encoding function presented in Fig. 25 that allows to combine two original expressions into a single extended expression. It is straightforward to check that binary encoding and projection have a natural correspondence, which fact we state as Lem. 17.

Lemma 17 (Projection of binary encoding).

$$\forall i \in \{1, 2\}.\ \lfloor v_1 \star v_2 \rfloor_i = v_i$$

Proof. Induction on the structure of v_1 and v_2. □

Next in Fig. 26 we present operational semantics for the extended language, parameterized by a branch ι where a computation takes place, and a table store $\mathbb{\Omega}$ mapping table names to extended relations, i.e., $T\{\overline{\overline{f : \mathbb{v}}}\}$. Branch ι can be either • for the top level, i.e., affecting both branches, 1 for the first branch, and 2 — for the second branch. Note, the definition and the structure of the context

$$\lfloor \mathbb{e} \rfloor_i = \begin{cases} e_i & \text{if } \mathbb{e} = \langle e_1 \mid e_2 \rangle \\ \lfloor \mathbb{e}_1 \rfloor_i, \ldots, \lfloor \mathbb{e}_n \rfloor_i & \text{if } \mathbb{e} = \overline{\mathbb{e}} = (\mathbb{e}_j)_{j \in \{1,\ldots,n\}} \\ \oplus(\lfloor \overline{\mathbb{e}} \rfloor_i) & \text{if } \mathbb{e} = \oplus(\overline{\mathbb{e}}) \\ \theta(\lfloor \overline{\mathbb{e}} \rfloor_i) & \text{if } \mathbb{e} = \theta(\overline{\mathbb{e}}) \\ \lfloor \mathbb{e}' \rfloor_i . f & \text{if } \mathbb{e} = \mathbb{e}'.f \\ \mathsf{encr}(\lfloor \mathbb{e}' \rfloor_i, s) & \text{if } \mathbb{e} = \mathsf{encr}(\mathbb{e}', s) \\ \mathsf{decr}(\lfloor \mathbb{e}' \rfloor_i) & \text{if } \mathbb{e} = \mathsf{decr}(\mathbb{e}') \\ [\lfloor \mathbb{e}' \rfloor_i]_d & \text{if } \mathbb{e} = [\mathbb{e}']_d \\ \lfloor \mathbb{e}' \rfloor_i(\lfloor \overline{e} \rfloor_i) & \text{if } \mathbb{e} = \mathbb{e}'(\overline{\mathbb{e}}) \\ \lambda[d](\overline{x : \kappa}).\ \lfloor \mathbb{e}' \rfloor_i & \text{if } \mathbb{e} = \lambda[d](\overline{x : \kappa}).\ \mathbb{e}' \\ \{\overline{f : \lfloor \mathbb{e} \rfloor_i}\} & \text{if } \mathbb{e} = \{\overline{f : \mathbb{e}}\} \\ T\{\overline{\overline{f : \lfloor \mathbb{v} \rfloor_i}}\} & \text{if } \mathbb{e} = T\{\overline{\overline{f : \mathbb{v}}}\} \\ \mathbb{e} & \text{if } \mathbb{e} = c \text{ or } \mathbb{e} = \mathtt{table}(name) \end{cases}$$

Fig. 24. Projection $\lfloor \mathbb{e} \rfloor_i$ of a term $\mathbb{e}$

$$v \star w = \begin{cases} v & \text{if } v = w \\ \{\overline{f : v \star w}\} & \text{if } v = \{\overline{f : v}\} \text{ and } w = \{\overline{f : w}\} \\ T\{\overline{\overline{f : v \star w}}\} & \text{if } v = \{\overline{\overline{f : v}}\} \text{ and } w = \{\overline{\overline{f : w}}\} \\ \langle v \mid w \rangle & \text{otherwise} \end{cases}$$

Fig. 25. Binary encoding

remains the same. We present operational rules pushing operations from the original language across the bracket construct in a separate figure, Fig. 27.

F.2 Properties of Extended Evaluation

In this part we show the soundness and completeness of evaluation relation $\underset{\Omega}{\Longrightarrow}$ with respect to original evaluation $\underset{\Omega}{\longrightarrow}$. Soundness means that $\underset{\Omega}{\Longrightarrow}$ does not introduce any bogus evaluation rules, and completeness essentially says that any terminating computation from the original language can be over to extended language. The two are connected by projection function $\lfloor\ \rfloor_i$.

F.2.1 Some technical lemmata Before we can actually attack the above properties, we introduce a projection of context C in Fig. 28.

And in addition, we use the following technical lemmata: Lem. 20 showing that projection distributes over substitution and Lem. 19 showing that projection distributes over putting an expression into a context.

Lemma 18 (Projection and encoding cancel). *For a non-function value* $\mathbb{v}$, $\mathbb{v} = \lfloor \mathbb{v} \rfloor_1 \star \lfloor \mathbb{v} \rfloor_1$.

Proof. Induction over structure of $\mathbb{v}$. □

Lemma 19 (Projection distributes over context application). *Let* $k \in$ *{1, 2}.* $\forall \mathbb{e}, \forall C$ *if* $\mathbb{e} = C[\mathbb{e}']$, *then* $\lfloor \mathbb{e} \rfloor_k = \lfloor C \rfloor_k[\lfloor \mathbb{e}' \rfloor_k]$.

Proof. By induction on the structure of $\mathbb{e}$ we consider the different shapes $\mathbb{e}$ can take. For each shape of $\mathbb{e}$ we look at its possible decomposition into a context whose hole is occupied with a subterm. We pick w.l.o.g. some $k \in \{1, 2\}$. The property we set out to prove,

$$\lfloor \mathbb{e} \rfloor_k = \lfloor C \rfloor_k[\lfloor \mathbb{e}' \rfloor_k] \tag{11}$$

In all cases, from the premise of the lemma we have

$$\mathbb{e} = C[\mathbb{e}'] \tag{12}$$

$$\text{(Ext-Ctx)}\ \frac{\mathbb{e}_1 \overset{\iota}{\underset{\Omega}{\Longrightarrow}} \mathbb{e}_2}{C[\mathbb{e}_1] \overset{\iota}{\underset{\Omega}{\Longrightarrow}} C[\mathbb{e}_2]} \qquad \text{(Ext-Op)}\ \frac{\varphi^{\mathrm{ev}}(\oplus, \overline{c, s}) = (c, s)}{\oplus(\overline{c^s}) \overset{\iota}{\underset{\Omega}{\Longrightarrow}} c^s}$$

$$\text{(Ext-Encr)}\ \frac{\varphi^{\mathrm{ev}}(\mathsf{encr}, c, s) = c_1 \qquad s \neq \varnothing}{\mathsf{encr}(c^{\varnothing}, s) \overset{\iota}{\underset{\Omega}{\Longrightarrow}} c_1^s} \qquad \text{(Ext-Decr)}\ \frac{\varphi^{\mathrm{ev}}(\mathsf{decr}, c, s) = c_1 \qquad s \neq \varnothing}{\mathsf{decr}(c^s) \overset{\iota}{\underset{\Omega}{\Longrightarrow}} c_1^{\varnothing}}$$

$$\text{(Ext-Tbl)}\ \frac{\Omega(name) = \mathbb{v}}{\mathtt{table}(name) \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \mathbb{v}} \qquad \text{(Ext-TblProj)}\ \frac{\Omega(name) = \mathbb{v} \qquad \iota \in \{1, 2\}}{\mathtt{table}(name) \overset{\iota}{\underset{\Omega}{\Longrightarrow}} \lfloor \mathbb{v} \rfloor_\iota} \qquad \text{(Ext-Return)}\ [\mathbb{v}]_d \overset{\iota}{\underset{\Omega}{\Longrightarrow}} \mathbb{v} \qquad \text{(Ext-RecSelect)}\ \{\overline{f : \mathbb{v}}\}.f_k \overset{\iota}{\underset{\Omega}{\Longrightarrow}} \mathbb{v}_k$$

$$\text{(Ext-OpQuery)}\ \frac{\overline{\mathbb{v}} = (\mathbb{v}_i)_{i \in I} \qquad \forall k \in I.\ \mathbb{v}_k \neq \langle T\{\overline{\overline{f : v_{k,1}}}\} \mid T\{\overline{\overline{f : v_{k,2}}}\}\rangle \qquad \varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_1) = v_1' \qquad \varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_2) = v_2'}{\theta(\overline{\mathbb{v}}) \overset{\iota}{\underset{\Omega}{\Longrightarrow}} v_1' \star v_2'}$$

$$\text{(Ext-Apply)}\ \lambda[d](\overline{x : \kappa}).\mathbb{e}(\overline{\mathbb{v}}) \overset{\iota}{\underset{\Omega}{\Longrightarrow}} [\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}]_d \qquad \text{(Ext-Bracket)}\ \frac{e_\iota \overset{\iota}{\underset{\Omega}{\Longrightarrow}} e_\iota' \qquad e_\zeta = e_\zeta' \qquad \{\iota, \zeta\} = \{1, 2\}}{\langle e_1 \mid e_2 \rangle \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \langle e_1' \mid e_2' \rangle}$$

Fig. 26. Translation of the original operational semantics to the extended language. In general $\iota \in \{\bullet, 1, 2\}$. The premises in rules (Ext-Op), (Ext-Encr), (Ext-Decr) hold only for "non-bracket" values, hence implicitly $\iota \in \{1, 2\}$. In (Ext-OpQuery), a "bracket" could appear at a non-root level.

$$\text{(Ext-LiftOp)}\ \frac{\overline{\mathbb{v}} = (\mathbb{v}_i)_{i \in I} \qquad \mathbb{v}_k = \langle v_{k1} \mid v_{k2} \rangle}{\oplus(\overline{\mathbb{v}}) \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \langle \oplus(\lfloor \overline{\mathbb{v}} \rfloor_1) \mid \oplus(\lfloor \overline{\mathbb{v}} \rfloor_2) \rangle} \qquad \text{(Ext-LiftOpQuery)}\ \frac{\overline{\mathbb{v}} = (\mathbb{v}_i)_{i \in I} \qquad \mathbb{v}_k = \langle T\{\overline{\overline{f : v_{k,1}}}\} \mid T\{\overline{\overline{f : v_{k,2}}}\}\rangle}{\theta(\overline{\mathbb{v}}) \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \langle \theta(\lfloor \overline{\mathbb{v}} \rfloor_1) \mid \theta(\lfloor \overline{\mathbb{v}} \rfloor_2) \rangle}$$

$$\text{(Ext-LiftEncr)}\ \frac{\mathbb{v} = \langle v_1 \mid v_2 \rangle \qquad s \neq \varnothing}{\mathsf{encr}(\mathbb{v}, s) \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \langle \mathsf{encr}(v_1, s) \mid \mathsf{encr}(v_2, s) \rangle} \qquad \text{(Ext-LiftDecr)}\ \frac{\mathbb{v} = \langle v_1 \mid v_2 \rangle}{\mathsf{decr}(\mathbb{v}) \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \langle \mathsf{decr}(v_1) \mid \mathsf{decr}(v_2) \rangle}$$

$$\text{(Ext-LiftRecSelect)}\ \langle \{\overline{f : v}\} \mid \{\overline{f : w}\} \rangle.f_k \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \langle \{\overline{f : v}\}.f_k \mid \{\overline{f : w}\}.f_k \rangle \qquad \text{(Ext-LiftApply)}\ \langle v_1 \mid v_2 \rangle(\overline{\mathbb{v}}) \overset{\bullet}{\underset{\Omega}{\Longrightarrow}} \langle v_1 \lfloor \overline{\mathbb{v}} \rfloor_1 \mid v_2 \lfloor \overline{\mathbb{v}} \rfloor_2 \rangle$$

Fig. 27. Lifting rules added to original operational semantics of the extended language.

From induction hypothesis for immediate subterms $\mathbb{e}_s$ of $\mathbb{e}$ it holds that

$$\begin{aligned} \forall \mathbb{e}_s, \forall C'.\ \lfloor \mathbb{e}_s \rfloor_k = \lfloor C'[\mathbb{e}_s'] \rfloor_k &= \lfloor C' \rfloor_k [\lfloor \mathbb{e}_s' \rfloor_k] \\ \textit{with} \qquad \mathbb{e}_s &= C'[\mathbb{e}_s'] \end{aligned} \tag{13}$$

- Case $\mathbb{e} = \oplus(\overline{\mathbb{e}})$
  Following sub-case arise in the decomposition of $\mathbb{e}$.
  - Sub-case $C = \oplus(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}})$
    We have $\mathbb{e} = C[\mathbb{e}'] = \oplus(\overline{\mathbb{v}}, \mathbb{e}', \overline{\mathbb{e}})$ from (12).

$$\lfloor C \rfloor_i = \begin{cases} [\bullet]_d & \text{if } C = [\bullet]_d \\ \oplus(\lfloor \overline{\mathbb{v}} \rfloor_i, \bullet, \lfloor \overline{\mathbb{e}} \rfloor_i) & \text{if } C = \oplus(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}}) \\ \theta(\lfloor \overline{\mathbb{v}} \rfloor_i, \bullet, \lfloor \overline{\mathbb{e}} \rfloor_i) & \text{if } C = \theta(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}}) \\ \mathsf{encr}(\bullet, s) & \text{if } C = \mathsf{encr}(\bullet, s) \\ \mathsf{decr}(\bullet) & \text{if } C = \mathsf{decr}(\bullet) \\ \bullet(\lfloor \overline{\mathbb{e}} \rfloor_i) & \text{if } C = \bullet(\overline{\mathbb{e}}) \\ \lfloor \mathbb{v} \rfloor_i(\lfloor \overline{\mathbb{v}} \rfloor_i, \bullet, \lfloor \overline{\mathbb{e}} \rfloor_i) & \text{if } C = \mathbb{v}(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}}) \\ \bullet.f & \text{if } C = \bullet.f \\ \{\overline{f : \lfloor \mathbb{v} \rfloor_i}, f : \bullet, \overline{f : \lfloor \mathbb{e} \rfloor_i}\} & \text{if } C = \{\overline{f : \mathbb{v}}, f : \bullet, \overline{f : \mathbb{e}}\} \end{cases}$$

Fig. 28. Projection $\lfloor C \rfloor_i$ of a context C

From Fig. 24

$$\lfloor \mathbb{e} \rfloor_k = \oplus(\lfloor \overline{\mathbb{v}} \rfloor_k, \lfloor \mathbb{e}' \rfloor_k, \lfloor \overline{\mathbb{e}} \rfloor_k) \tag{14}$$

From Fig. 28, 14, and $C' = \oplus(\lfloor \overline{\mathbb{v}} \rfloor_k, \bullet, \lfloor \overline{\mathbb{e}} \rfloor_k)$ we have

$$\lfloor \mathbb{e} \rfloor_k = C'[\lfloor \mathbb{e}' \rfloor_k] \tag{15}$$

The result - (11) follows from noting that $C' = \lfloor C \rfloor_k$ due to Fig. 28.

- Case $\mathbb{e} = \theta(\overline{\mathbb{e}})$ is similar to the previous case.
- Case $\mathbb{e} = \mathbb{v}$ (i.e., $\mathbb{e} = x$, $\mathbb{e} = c^s$, $\mathbb{e} = f$, $\mathbb{e} = T\{\overline{\overline{f : \mathbb{v}}}\}$, $\mathbb{e} = \{\overline{f : \mathbb{v}}\}$, $\mathbb{e} = \lambda[d](\overline{x : \kappa}).\ \mathbb{e}$, ...) cannot be decomposed into a context whose hole is filled with a subterm.
- Case $\mathbb{e} = [\mathbb{e}']_d$
  - Sub-case $C = [\bullet]_d$
    We have $\mathbb{e} = [\mathbb{e}']_d$. From Fig. 24 and Fig. 28

$$\lfloor \mathbb{e} \rfloor_k = \lfloor [\mathbb{e}']_d \rfloor_k = [\lfloor \mathbb{e}' \rfloor_k]_d = C'[\lfloor \mathbb{e}' \rfloor_k] = \lfloor C \rfloor_k[\lfloor \mathbb{e}' \rfloor_k] \tag{16}$$

- Case $\mathbb{e} = \mathbb{e}_\lambda(\overline{\mathbb{e}})$
  - Sub-case $C = \mathbb{v}(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}})$
    From Fig. 24 and Fig. 28

$$\lfloor \mathbb{e} \rfloor_k = \lfloor \mathbb{e}_\lambda(\overline{\mathbb{e}}) \rfloor_k = \lfloor \mathbb{e}_\lambda(\overline{\mathbb{v}}, \mathbb{e}', \overline{\mathbb{e}}) \rfloor_k = \lfloor \mathbb{e}_\lambda \rfloor_k(\lfloor \overline{\mathbb{v}} \rfloor_k, \lfloor \mathbb{e}' \rfloor_k, \lfloor \overline{\mathbb{e}} \rfloor_k) = \lfloor C \rfloor_k[\lfloor \mathbb{e}' \rfloor_k] \tag{17}$$

  - Sub-case $C = \bullet(\overline{\mathbb{e}})$ is similar to the above sub-case.
- Case $\mathbb{e} = \{\overline{f : \mathbb{e}}\}$
  - $C = \{\overline{f : \mathbb{v}}, f : \bullet, \overline{f : \mathbb{e}}\}$. Straightforward from Fig. 24 and Fig. 28.
- Case $\mathbb{e} = \mathbb{e}'.f$. Straightforward from Fig. 24 and Fig. 28.
  - Sub-case $C = \bullet.f$
- Case $\mathbb{e} = \mathsf{encr}(\mathbb{e}', s)$
  - Sub-case $C = \mathsf{encr}(\bullet, s)$
    We have $\mathbb{e} = \mathsf{encr}(\mathbb{e}', s)$. From Fig. 24

$$\lfloor \mathbb{e} \rfloor_k = \mathsf{encr}(\lfloor \mathbb{e}' \rfloor_k, s) = C'[\lfloor \mathbb{e}' \rfloor_k] = \lfloor C \rfloor_k[\lfloor \mathbb{e}' \rfloor_k] \tag{18}$$

- Case $\mathbb{e} = \mathsf{decr}(\mathbb{e}')$ is similar to the previous case.

□

Lemma 20 (projection distributes over substition).

$$\forall i \in \{1, 2\}.\ \lfloor \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} \rfloor_i = \{\lfloor \overline{\mathbb{v}} \rfloor_i/\overline{x}\}\lfloor \mathbb{e} \rfloor_i$$

Proof. Induction over the structure of $\mathbb{e}$. □

F.2.2 *Soundness*

Theorem 6 (Soundness of extended evaluation). *If* $\mathbb{e} \xRightarrow[\Omega]{\iota} \mathbb{e}'$ *then for any* $i \in \{1,2\}$ *either* $\lfloor \mathbb{e} \rfloor_i \xrightarrow[\lfloor \Omega \rfloor_i]{} \lfloor \mathbb{e}' \rfloor_i$ *or* $\lfloor \mathbb{e} \rfloor_i = \lfloor \mathbb{e}' \rfloor_i$.

Proof. By an induction on the final evaluation rule in the derivation of $\mathbb{e} \xrightarrow[\Omega]{} \mathbb{e}'$. We proceed by case analysis on the rules from Fig. 26.
From the premise of the lemma, in all cases we have

$$\mathbb{e} \xRightarrow[\Omega]{\iota} \mathbb{e}' \tag{19}$$

We pick w.l.o.g. some $i \in \{1,2\}$ for showing that either $\lfloor \mathbb{e} \rfloor_i \xrightarrow[\lfloor \Omega \rfloor_i]{} \lfloor \mathbb{e}' \rfloor_i$ or $\lfloor \mathbb{e} \rfloor_i = \lfloor \mathbb{e}' \rfloor_i$ holds.

- Case (Ext-Op). From the premise of (Ext-Op) it holds that
$$\overline{v} = (v_i)_{i \in I} \qquad \forall k \in I.\ v_k \neq \langle v_{k1} \mid v_{k2} \rangle \qquad \varphi^O(\oplus, \overline{v}) = v' \tag{20}$$
From Fig. 24 and (20), we have
$$\lfloor \oplus(\overline{v}) \rfloor_i = \oplus(\lfloor \overline{v} \rfloor_i) = \oplus(\overline{v}) \qquad \lfloor v' \rfloor_i = v' \tag{21}$$
From (20), (21) and (Ev Op) we have that $\lfloor \oplus(\overline{v}) \rfloor_i \xrightarrow[\Omega]{} \lfloor v' \rfloor_i$.
- Case (Ext-OpQuery). From the premise of (Ext-OpQuery) it holds that
$$\begin{gathered} \overline{\mathbb{v}} = (\mathbb{v}_i)_{i \in I} \qquad \forall k \in I.\ \mathbb{v}_k \neq \langle v_{k1} \mid v_{k2} \rangle \\ \varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_1) = v_1' \qquad \varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_2) = v_2' \end{gathered} \tag{22}$$
Hence from (Ev OpQuery) we know that for any $i \in \{1,2\}$: $\theta(\lfloor \overline{\mathbb{v}} \rfloor_i) \xrightarrow[\lfloor \Omega \rfloor_i]{i} v_i'$
By definition of $\lfloor \_ \rfloor_i$ and Lem. 17 we also have for any $i \in \{1,2\}$:
$$\lfloor \theta(\overline{\mathbb{v}}) \rfloor_i = \theta(\lfloor \overline{\mathbb{v}} \rfloor_i) \qquad \lfloor v_1' \star v_2' \rfloor_i = v_i' \tag{23}$$
And the conclusion follows both for $i = 1$ and $i = 2$ using (Ev OpQuery).
- Cases (Ext-Encr), (Ext-Decr) are similar to case (Ext-Op).
- Case (Ext-Apply). By the definition of projection:
$$\lfloor \lambda[d](\overline{x : \kappa}).\mathbb{e}(\overline{\mathbb{v}}) \rfloor_i = \lambda[d](\overline{x : \kappa}).\lfloor \mathbb{e} \rfloor_i(\lfloor \overline{\mathbb{v}} \rfloor_i) \tag{24}$$
By the definition of projection Fig. 24 and Lem. 20
$$\lfloor [\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}]_d \rfloor_i = [\lfloor \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} \rfloor_i]_d = [\{\lfloor \overline{\mathbb{v}} \rfloor_i/\overline{x}\}\lfloor \mathbb{e} \rfloor_i]_d \tag{25}$$
Hence, by (Ev Apply) the conclusion holds for both $i = 1$ and $i = 2$.
- Case (Ext-RecSelect). From Fig. 24, we have
$$\begin{aligned} \lfloor \{\overline{f : \mathbb{v}}\}.f_j \rfloor_i &= \{\overline{f : \lfloor \mathbb{v} \rfloor_i}\}.f_j \\ = \{f_1 : \lfloor \mathbb{v}_1 \rfloor_i, \ldots, f_j : \lfloor \mathbb{v}_j \rfloor_i, \ldots, f_n : \lfloor \mathbb{v}_n \rfloor_i\}.f_j &= \lfloor \mathbb{v}_j \rfloor_i \end{aligned} \tag{26}$$
Hence, when the last rule applied in the evaluation derivation is (Ext-RecSelect), we have that $\lfloor \mathbb{e} \rfloor_i = \lfloor \mathbb{e}' \rfloor_i$.
- Case (Ext-LiftOp). From (19) we have
$$\oplus(\overline{\mathbb{v}}) \xRightarrow[\Omega]{\bullet} \langle \oplus(\lfloor \overline{\mathbb{v}} \rfloor_1) \mid \oplus(\lfloor \overline{\mathbb{v}} \rfloor_2) \rangle \tag{27}$$
From Fig. 24, we have
$$\lfloor \oplus(\overline{\mathbb{v}}) \rfloor_i = \oplus(\lfloor \overline{\mathbb{v}} \rfloor_i) \qquad \lfloor \langle \oplus(\lfloor \overline{\mathbb{v}} \rfloor_1) \mid \oplus(\lfloor \overline{\mathbb{v}} \rfloor_2) \rangle \rfloor_i = \oplus(\lfloor \overline{\mathbb{v}} \rfloor_i) \tag{28}$$

From (28), we have that

$$\lfloor \mathbb{e} \rfloor_i = \lfloor \mathbb{e}' \rfloor_i \tag{29}$$

- Cases (Ext-LiftOpQuery), (Ext-LiftEncr), (Ext-LiftDecr), (Ext-LiftRecSelect), (Ext-LiftApply) are similar to case (Ext-LiftOp).
- Case (Ext-Ctx). That is the last rule used in the evaluation derivation is,

$$C[\mathbb{e}_1] \overset{\iota}{\underset{\Omega}{\Longrightarrow}} C[\mathbb{e}_2] \tag{30}$$

By an inversion on rule (Ext-Ctx), we have

$$\mathbb{e}_1 \overset{\iota}{\underset{\Omega}{\Longrightarrow}} \mathbb{e}_2 \tag{31}$$

And from induction hypothesis on (31), we have

$$\textit{Either} \qquad \lfloor \mathbb{e}_1 \rfloor_i \xrightarrow[\lfloor \Omega \rfloor_i]{} \lfloor \mathbb{e}_2 \rfloor_i \qquad \textit{or} \qquad \lfloor \mathbb{e}_1 \rfloor_i = \lfloor \mathbb{e}_2 \rfloor_i \tag{32}$$

From Lem. 19, we have

$$\lfloor C[\mathbb{e}_1] \rfloor_i = \lfloor C \rfloor_i [\lfloor \mathbb{e}_1 \rfloor_i] \qquad \lfloor C[\mathbb{e}_2] \rfloor_i = \lfloor C \rfloor_i [\lfloor \mathbb{e}_2 \rfloor_i] \tag{33}$$

The result follows from (32), (33) and (Ev Cxt) with context as $\lfloor C \rfloor_i$.
- Case (Ext-Bracket). From inversion on (Ext-Bracket), conclusion is straightforward from the premises.

□

F.2.3 Completeness For completeness we first show in Lem. 21 that extended evaluation does not get stuck unless one of the branches are stuck.

Lemma 21 (Enough lifting rules). *If $\mathbb{e}$ is stuck wrt $\underset{\Omega}{\Longrightarrow}$, then $\lfloor \mathbb{e} \rfloor_i$ is stuck wrt $\xrightarrow[\lfloor \Omega \rfloor_i]{}$ for some $i \in \{1, 2\}$.*

Proof. By induction on structure of $\mathbb{e}$.

- Case $\mathbb{e} = T\{\overline{\overline{f : \mathbb{v}}}\} \mid \{\overline{f : \mathbb{v}}\} \mid \lambda[d](\overline{x : \kappa}).\, \mathbb{e} \mid \langle v \mid v \rangle \mid v$
  $\mathbb{e}$ is a value, hence $\mathbb{e}$ is not stuck. Therefore Lem. 21 holds.
- Case $\mathbb{e} = \mathbb{e}'(\mathbb{e}'')$.
  As $\mathbb{e}$ is stuck, (Ext-Apply) is not applicable. Hence, $\mathbb{e}' \neq \lambda[d](\overline{x : \kappa}).\, \mathbb{e}_b$ and from Fig. 24 we have $\lfloor \mathbb{e}' \rfloor_i \neq \lambda[d](\overline{x : \kappa}).\, \lfloor \mathbb{e}_b \rfloor_i$ and $\lfloor \mathbb{e} \rfloor_i \neq \lambda[d](\overline{x : \kappa}).\, \lfloor \mathbb{e}_b \rfloor_i(\mathbb{e}'')$ for $i \in \{1, 2\}$. Hence, $\lfloor \mathbb{e} \rfloor_i$ is also stuck.
  (Ext-LiftApply) is not applicable as $\mathbb{e}$ is stuck and hence $\mathbb{e}' \neq \langle \mathbb{e}_1 \mid \mathbb{e}_2 \rangle$. Consequently, $\lfloor \mathbb{e}' \rfloor_i \neq \mathbb{e}_i$ and $\lfloor \mathbb{e} \rfloor_i \neq \mathbb{e}_i(\mathbb{e}'')$ for $i \in \{1, 2\}$. Hence, $\lfloor \mathbb{e} \rfloor_i$ is also stuck.
- Case $\mathbb{e} = \theta(\overline{\mathbb{v}})$.
  – (Ext-LiftOpQuery) is not applicable $\Rightarrow \forall i.\, v_i \neq \langle \_, \_ \rangle$
  – $\forall i.\, v_i \neq \langle \_, \_ \rangle \wedge$ (Ext-OpQuery) is not applicable $\Rightarrow \varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_1) = \bot \vee \varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_2) = \bot$
  – Wlog $\varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_1) = \bot \Rightarrow \lfloor \mathbb{e} \rfloor_1 = \lfloor \theta(\overline{\mathbb{v}}) \rfloor_1 = \theta(\lfloor \overline{\mathbb{v}} \rfloor_1)$ is stuck.
- Case $\mathbb{e} = \oplus(\overline{\mathbb{v}})$.
  By contradiction. If both $\oplus(\lfloor \overline{\mathbb{v}} \rfloor_1)$ and $\oplus(\lfloor \overline{\mathbb{v}} \rfloor_2)$ are not stuck, then rule (Ev Op) applies for both projection, then, by Fig. 24, there are only two subcases:
  – $\overline{\mathbb{v}} = \overline{v}$ and $\varphi^{\mathrm{ev}}(\oplus, \overline{v}) = v'$, where (Ext-Op) applies; or
  – there exists i, s.t. $\mathbb{v}_i = \langle v_{i1} \mid v_{i2} \rangle$, where (Ext-LiftOp) applies.
- Case $\mathbb{e} = \mathbb{v}.f$.
  By contradiction. If both $\lfloor \mathbb{v} \rfloor_1.f$ and $\lfloor \mathbb{v} \rfloor_2.f$ are not stuck, then rule (Ev RecSelect) applies for both projection, then, by Fig. 24, there are only two subcases:

  - $\mathbb{v} = \{\overline{f : \mathbb{v}}\}$ and $f = f_k$, where (Ext-RecSelect) applies; or
  - $\mathbb{v} = \langle \overline{f : v_1} \mid \overline{f : v_2} \rangle$ and $f = f_k = f_j$, where (Ext-LiftRecSelect) applies.
- Case $\mathbb{e} = \texttt{encr}(\mathbb{v}, s)$ where $s \neq \emptyset$. By contradiction. If both $\texttt{encr}(\lfloor \mathbb{v} \rfloor_1, s)$ and $\texttt{encr}(\lfloor \mathbb{v} \rfloor_2, s)$ are not stuck, then (Ev Enc) applies for both projection, then, by Fig. 24, there are only two subcases:
  - $\mathbb{v} = c$ and $\varphi^{\text{ev}}(\texttt{encr}, c, s) = c^s$, where (Ext-Encr) applies.
  - $\mathbb{v} = \langle c_1 \mid c_2 \rangle$, where (Ext-LiftEncr) applies.
- Case $\mathbb{e} = \texttt{decr}(\mathbb{v})$ where $s \neq \emptyset$.
  By contradiction. If both $\texttt{decr}(\lfloor \mathbb{v} \rfloor_1)$ and $\texttt{decr}(\lfloor \mathbb{v} \rfloor_2)$ are not stuck, then (Ev Decr) applies for both projection, then, by Fig. 24, there are only two subcases:
  - $\mathbb{v} = c^s$ and $\varphi^{\text{ev}}(\texttt{decr}, c^s) = c$, where (Ext-Decr) applies.
  - $\mathbb{v} = \langle c_1^s \mid c_2^s \rangle$, where (Ext-LiftDecr) applies.
- Case $\mathbb{e} = \texttt{table}(\mathit{name})$.
  Since (Ext-Tbl) does not apply *name* is not in $\mathit{dom}(\Omega)$, hence *name* is not in any of $\mathit{dom}(\lfloor \Omega \rfloor_i)$ and (Ev Tbl) does not apply.
- Case $\mathbb{e} = [\mathbb{v}]_d$.
  Impossible as (Ext-Return) applies.
- Case $\mathbb{e} = C[\mathbb{e}_1]$.
  As $\mathbb{e}$ is stuck, it should be the case that $\mathbb{e}_1$ is stuck, lest $\mathbb{e}$ can take a step of evaluation by (Ext-Ctx).
  For $\mathbb{e}_1$ to be a subterm of $\mathbb{e}$, it should hold that $C \neq [\ ]$ (i.e., C is not an empty context). Then, induction hypothesis is applicable to $\mathbb{e}_1$ - a subterm of $\mathbb{e}$, and thus we have
  $$\lfloor \mathbb{e}_1 \rfloor_i \text{ is stuck for some } i \in \{1, 2\} \tag{34}$$
  From Lem. 19, we have
  $$\lfloor \mathbb{e} \rfloor_i = \lfloor C[\mathbb{e}_1] \rfloor_i = \lfloor C \rfloor_i [\lfloor \mathbb{e}_1 \rfloor_i] \tag{35}$$
  By inspection of the reduction rules in Fig. 26, we notice only (Ext-Ctx) concerns itself with evaluation of non normal subterms enclosed in a non-empty context. Other rules evaluate expressions with either subterms in normal form enclosed in a non-empty context or subterms in non normal form enclosed in a $\langle \cdot \mid \cdot \rangle$ construct.
  Hence, by (34) and (Ext-Ctx), for all C', $C'[\lfloor \mathbb{e}_1 \rfloor_i]$ is stuck for some $i \in \{1, 2\}$; in particular, for $C' = \lfloor C \rfloor_i$ the expression in (35) stuck.
- Case $\mathbb{e} = \langle e_1 \mid e_2 \rangle$.
  From the premise of (Ext-Bracket) it must be the case that e_1 and e_2 are stuck for $\mathbb{e}$ to be stuck. Noting that $\lfloor \mathbb{e} \rfloor_i = e_i$, the lemma holds.

□

Theorem 7 (Completeness of extended language evaluation). *If for all $i \in \{1, 2\}$ $\lfloor \mathbb{e} \rfloor_i \xrightarrow[\lfloor \Omega \rfloor_i]{} v_i$ then there exists $\mathbb{v}$ such that $\mathbb{e} \underset{\Omega}{\Longrightarrow}^* \mathbb{v}$, and for all $j \in \{1, 2\}$, $\lfloor \mathbb{v} \rfloor_j = v_j$.*

Proof. We first establish that terms of the extended language do not admit an infinite evaluation sequence. Due to Th. 6 the image under projection of a valid evaluation sequence in the extended language becomes a valid evaluation sequence in the original language if consecutive equal elements are removed. It is straightforward to show by the induction on evaluation relation Fig. 26 that consecutive equal elements are precisely the reductions involving "lift" rules. No infinite reduction sequence can consist purely of "lift" reduction rules as each "lift" rule moves the $\langle \cdot \mid \cdot \rangle$ construction closer to term's root, hence an infinite evaluation sequence remains infinite after removing repeated

$$\text{(Ext-T-TblCall)}\quad \frac{\begin{array}{c}\rho(name) = T\{\overline{f : (p^s, l)}\} \\ \forall i.\ (p_i^s, l_i) \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \texttt{table}(name) : T\{\overline{f : (p^s, l)}\}}$$

$$\text{(Ext-T-Tbl)}\quad \frac{\begin{array}{c}\forall j.\ \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v}_{i,j} : (p_i^s, l_i)} \\ \forall i.\ (p_i^s, l_i) \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} T\{\overline{\overline{f_i : \mathbb{v}_{i,j}}^{\,i}}^{\,j}\} : T\{\overline{f : (p^s, l)}\}}$$

$$\text{(Ext-T-Const)}\quad \frac{\varphi^{\text{ty}}(c, s) = p^s}{\rho \wr \Gamma \Vdash_{d/d_0} c^s : (p^s, \bot)}$$

$$\text{(Ext-T-Record)}\quad \frac{\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : (p^s, l)}}{\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{f : \mathbb{e}}\} : \{\overline{f : (p^s, l)}\}}$$

$$\text{(Ext-T-RecSelect)}\quad \frac{\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{f : \mathbb{e}}\} : \{\overline{f : (p^s, l)}\}}{\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}.f_i : (p_i^s, l_i)}$$

$$\text{(Ext-T-Var)}\quad \frac{\begin{array}{c}\Gamma(x) = \kappa \\ \kappa \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} x : \kappa}$$

$$\text{(Ext-T-Fun)}\quad \frac{\begin{array}{c}\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{\tilde{d}/d_0} \mathbb{e} : \kappa \\ \forall i.\ \kappa_i \sqsubseteq \tilde{d}\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \lambda[\tilde{d}](\overline{x : \kappa}).\ \mathbb{e} : \overline{\kappa} \longrightarrow_{\tilde{d}} \kappa}$$

$$\text{(Ext-T-Return)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{\tilde{d}/d_0} \mathbb{e} : \kappa \\ \kappa \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} [\mathbb{e}]_{\tilde{d}} : \kappa}$$

$$\text{(Ext-T-ConfUp)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa \\ \kappa <: \tilde{\kappa} \\ \tilde{\kappa} \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \tilde{\kappa}}$$

$$\text{(Ext-T-Apply)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda : \overline{\kappa} \longrightarrow_{\tilde{d}} \kappa \\ \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : \kappa} \\ \kappa \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda(\overline{\mathbb{e}}) : \kappa}$$

$$\text{(Ext-T-Op)}\quad \frac{\begin{array}{c}\varphi^{\text{ty}}(\oplus) = \overline{p^{\bar{s}}} \longrightarrow p^s \\ s, \bar{s} \in \{s', \varnothing\} \\ \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : (p^s, l)} \\ (p^s, \sqcup_i l_i) \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \oplus(\overline{\mathbb{e}}) : (p^s, \sqcup_i l_i)}$$

$$\text{(Ext-T-Decr)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (p^s, l) \\ \varphi^{\text{ty}}(\mathsf{decr}) = p^s \longrightarrow p \\ (p, l) \sqsubseteq d \qquad s \neq \varnothing\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \mathsf{decr}(\mathbb{e}) : (p, l)}$$

$$\text{(Ext-T-Encr)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (p^{\varnothing}, l) \\ \varphi^{\text{ty}}(\mathsf{encr}) = p \longrightarrow p^s \\ (p^s, l) \sqsubseteq d \qquad s \neq \varnothing\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \mathsf{encr}(\mathbb{e}, s) : (p^s, l)}$$

Fig. 29. Typing judgements for the extended language: function declaration and application, encryption and decryption, etc.

elements. However, from the assumption of the current lemma, the term from the original language does evaluate to a final value in a finite number of evaluation steps. Thus we have a contradiction and hence a term of the extended language cannot admit an infinite evaluation sequence.

The possibility remains that a term from the extended language is stuck due to lack of appropriate lifting rules. By Lem. 21, the term's projection is also stuck. However, this contradicts the assumption of the current lemma that the term's projection evaluates to a value. □

F.3 Subject Reduction

F.3.1 Several technical properties of typing

Lemma 22 (Projection preserves typing). *If $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa$ then $\rho \wr \Gamma \Vdash_{d/d_0} \lfloor \mathbb{e} \rfloor_i : \kappa$*

Proof. By induction on the derivation of $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa$ expanding projection with Fig. 24. When $\mathbb{e}$ is a binary term, by inversion on the typing rules (Ext-T-Bracket), (Ext-T-Bracket-Enc), and (Ext-T-Bracket-Tbl) it holds that $\rho \wr \Gamma \vdash_d e_1 : \kappa$ and $\rho \wr \Gamma \vdash_d e_2 : \kappa$, and the claim follows. □

Lemma 23 (Type preservation across domains). *If $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa$ and $\kappa \sqsubseteq d'$ then $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e} : \kappa$, i.e., the type of an expression is preserved across compatible domains.*

Proof. By induction on the derivation of $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa$. We proceed by case analysis on the final typing rule in the derivation. In the induction hypothesis, we assume that the desired type preservation across domains property holds for all subderivations.

$$\text{(Ext-T-Filter)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i\in I} \qquad \rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda : \{f_i : (p_i^s, l_i)\}_{i\in I'} \longrightarrow_{d'} (\texttt{Bool}, l)\\ I' \subseteq I \qquad \forall i.\ (p_i^s, l_i \sqcup l) \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \texttt{filter}(\mathbb{e}_t, \mathbb{e}_\lambda) : T\{f_i : (p_i^s, l_i \sqcup l)\}_{i\in I}}$$

$$\text{(Ext-T-Cross)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_1 : T\{f_i : (p_i^s, l_i)\}_{i\in I} \qquad \rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_2 : T\{f_j : (p_j^s, l_j)\}_{j\in J}\\ J \cap I = \emptyset \qquad \forall k \in I \cup J.(p_k^s, l_k \sqcup (\sqcap_{i\in I} l_i) \sqcup (\sqcap_{j\in J} l_j)) \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \texttt{cross}(\mathbb{e}_1, \mathbb{e}_2) : T\{f_k : (p_k^s, l_k \sqcup (\sqcap_{i\in I} l_i) \sqcup (\sqcap_{j\in J} l_j))\}_{k\in I\cup J}}$$

$$\text{(Ext-T-Proj)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i\in I} \qquad I' \subseteq I\\ \rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda : \{f_i : (p_i^s, l_i)\}_{i\in I'} \longrightarrow_{d'} \{f_j : (p_j^s, l_j)\}_{j\in J} \qquad \forall j \in J.\ (p_j^s, l_j \sqcup (\sqcap_{i\in I} l_i)) \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \texttt{proj}(\mathbb{e}_t, \mathbb{e}_\lambda) : T\{f_j : (p_j^s, l_j \sqcup (\sqcap_{i\in I} l_i))\}_{j\in J}}$$

$$\text{(Ext-T-Agg)}\quad \frac{\begin{array}{c}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i\in I} \qquad \rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_0 : (p^s, l') \qquad I' \subseteq I \qquad j \in I\\ \rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda : (\{f_i : (p_i^s, l_i)\}_{i\in I'}, (p^s, l')) \longrightarrow_{d'} (p^s, l') \qquad (p^s, l' \sqcup l_j) \sqsubseteq d\end{array}}{\rho \wr \Gamma \Vdash_{d/d_0} \texttt{agg}(\mathbb{e}_t, f_j, \mathbb{e}_0, \mathbb{e}_\lambda) : T\{\texttt{key} : (p_j^s, l_j), \texttt{aggVal} : (p^s, l' \sqcup l_j)\}}$$

Fig. 30. Typing judgements for the extended language: query operators filter, join (i.e., cross-product), project, and aggregate.

In all cases, from the assumption of the lemma, we have

$$\begin{gathered}\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa\\ \kappa \sqsubseteq d'\end{gathered} \tag{36}$$

- Case (Ext-T-Const) is immediate since from the conclusion it holds that $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (\_, \bot)$. We know that $\forall d'.\ \bot \sqsubseteq d'$.
  Hence, $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e} : (\_, \bot)$.
- Case (Ext-T-Tbl). By inversion on (Ext-T-Tbl), we know that

$$\forall i.\ (p_i^s, l_i) \sqsubseteq d \tag{37}$$

  From induction hypothesis on the first premise of (Ext-T-Tbl), we know that

$$\forall j.\ \rho \wr \Gamma \Vdash_{d'/d_0} \overline{\mathbb{v}_{i,j} : (p_i^s, l_i)} \tag{38}$$

  From Equation 38, Equation 37, and (Ext-T-Tbl), the result follows.
- Case (Ext-T-Var). By inversion on (Ext-T-Var), we have

$$x : \kappa \in \Gamma \text{ and } \kappa \sqsubseteq d \tag{39}$$

  From (36), (39), and (Ext-T-Var) we have $\rho \wr \Gamma \Vdash_{d'/d_0} x : \kappa$.
- Case (Ext-T-Op). From (36), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \oplus(\overline{\mathbb{e}}) : (p^s, \sqcup l_i) \qquad \sqcup\, l_i \sqsubseteq d' \tag{40}$$

  By inversion on (Ext-T-Op), we have

$$\varphi^T = \overline{p^s} \longrightarrow p^s \qquad \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : (p^s, l)} \qquad \sqcup\, l_i \sqsubseteq d \tag{41}$$

  From induction hypothesis, we have $\rho \wr \Gamma \Vdash_{d'/d_0} \overline{\mathbb{e} : (p^s, l)}$. Using this, (40), (41), and (Ext-T-Op) we have $\rho \wr \Gamma \Vdash_{d'/d_0} \oplus(\overline{\mathbb{e}}) : (p^s, \sqcup l_i)$.
- Case (Ext-T-ConfUp). From (36), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \tilde{\kappa} \qquad \tilde{\kappa} \sqsubseteq d' \tag{42}$$

By inversion on (Ext-T-ConfUp), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa \qquad \kappa <: \tilde{\kappa} \qquad \tilde{\kappa} \sqsubseteq d \tag{43}$$

From induction hypothesis, we have $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e} : \kappa$. This along with $\tilde{\kappa} \sqsubseteq d'$, $\kappa <: \tilde{\kappa}$, and (Ext-T-ConfUp) gives us the result.

- Case (Ext-T-Fun) is similar to case (Ext-T-Tbl) and case (Ext-T-Const).
- Case (Ext-T-Apply). From (36), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda(\overline{\mathbb{e}}) : \kappa \tag{44a}$$
$$\kappa \sqsubseteq d' \tag{44b}$$

By inversion on (Ext-T-Apply), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda : \overline{\kappa} \longrightarrow_{\tilde{d}} \kappa \tag{45a}$$
$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : \kappa} \tag{45b}$$
$$\kappa \sqsubseteq d \tag{45c}$$

From induction hypothesis on (45a) and (45b), we have

$$\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}_\lambda : \overline{\kappa} \longrightarrow_{\tilde{d}} \kappa \qquad \rho \wr \Gamma \Vdash_{d'/d_0} \overline{\mathbb{e} : \kappa} \tag{46}$$

Using (44b), (46), and (Ext-T-Apply) we have $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}_\lambda(\overline{\mathbb{e}}) : \kappa$.

- Case (Ext-T-Decr). From (36), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathsf{decr}(\mathbb{e}) : (p, l) \tag{47a}$$
$$(p, l) \sqsubseteq d' \tag{47b}$$

By inversion on (Ext-T-Decr), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (p^s, l) \tag{48a}$$
$$\varphi^{\mathrm{ty}}(\mathsf{decr}) = p^s \longrightarrow p \tag{48b}$$
$$(p, l) \sqsubseteq d \tag{48c}$$
$$s \neq \varnothing \tag{48d}$$

From induction hypothesis on (48a), we have

$$\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e} : (p^s, l) \tag{49}$$

From (49), (48b), (48d), (47b), and (Ext-T-Decr) we have $\rho \wr \Gamma \Vdash_{d'/d_0} \mathsf{decr}(\mathbb{e}) : (p, l)$

- Case (Ext-T-Encr). From (36), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathsf{encr}(\mathbb{e}, s) : (p^s, l) \tag{50a}$$
$$(p^s, l) \sqsubseteq d' \tag{50b}$$

By inversion on (Ext-T-Encr), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (p^\varnothing, l) \tag{51a}$$
$$\varphi^{\mathrm{ty}}(\mathsf{encr}) = p \longrightarrow p^s \tag{51b}$$
$$(p^s, l) \sqsubseteq d \tag{51c}$$
$$s \neq \varnothing \tag{51d}$$

From induction hypothesis on (51a), we have

$$\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e} : (p^\varnothing, l) \tag{52}$$

From (52), (51b), (51d), (50b), and (Ext-T-Encr) we have $\rho \wr \Gamma \Vdash_{d'/d_0} \mathsf{encr}(\mathbb{e}, s) : (p^s, l)$.

- Case (Ext-T-Record). From (36), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{f : \mathbb{e}}\} : \{\overline{f : (p^s, l)}\} \qquad l \sqsubseteq d' \tag{53}$$

By inversion on (Ext-T-Record), we have

$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : (p^s, l)} \tag{54}$$

From induction hypothesis on (54), we have

$$\rho \wr \Gamma \Vdash_{d'/d_0} \overline{\mathbb{e} : (p^s, l)} \tag{55}$$

From (55), and (Ext-T-Record) we have $\rho \wr \Gamma \Vdash_{d'/d_0} \{\overline{f : \mathbb{e}}\} : \{\overline{f : (p^s, l)}\}$

- Case (Ext-T-RecSelect). Similar to case (Ext-T-Record).

□

Lemma 24 (Well-typedness preserved within context). *If $\rho \wr \Gamma \Vdash_{d/d_0} C[\mathbb{e}] : \kappa$ for some $\mathbb{e}$, then there exist κ' and d', s.t., for all $\mathbb{e}'$ we have $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}' : \kappa' \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0} C[\mathbb{e}'] : \kappa$.*

Proof. Proceed by induction on different shapes of C (see Fig. 4) while doing case analysis on $\rho \wr \Gamma \Vdash_{d/d_0} C[\mathbb{e}] : \kappa$ (see Fig. 29). For simplicity we assume that $\rho \wr \Gamma \Vdash_{d/d_0} C[\mathbb{e}] : \kappa$ does not include instances of (Ext-T-ConfUp), which can easily be addressed with the inner induction on the number of (Ext-T-ConfUp) at the root. In all cases, by assumption of the lemma

$$\rho \wr \Gamma \Vdash_{d/d_0} C[\mathbb{e}] : \kappa \tag{56}$$

- For $C = \oplus(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}})$, by Fig. 4, we have $C[\mathbb{e}] = \oplus(\overline{\mathbb{v}}, \mathbb{e}, \overline{\mathbb{e}})$.
  The only rule that matches $C[\mathbb{e}]$'s shape is (T-Op).
  By the inversion of (Ext-T-Op): (1) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_i : (p_i^s, l_i)$ for $1 \le i \le |\overline{\mathbb{v}}|$; (2) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_i : (p_i^s, l_i)$ for $|\overline{\mathbb{v}}| + 2 \le i \le |\overline{\mathbb{v}}| + 1 + |\overline{\mathbb{e}}|$; (3) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (p_k^s, l_k)$, where $k = |\overline{\mathbb{v}}| + 1$; (4) $\varphi^{\text{ty}}(\oplus) = \overline{p^s} \longrightarrow p^s$; and (5) $(p^s, \sqcup_i l_i) \sqsubseteq d$.
  Applying induction hypothesis to (3) we get some κ' and d', s.t. for all $\mathbb{e}'$ $\rho \wr \Gamma \vdash_{d'} \mathbb{e}' : \kappa' \Leftrightarrow \rho \wr \Gamma \vdash_d C'[\mathbb{e}'] : (p_k^s, l_k)$.
  It remains to show that for all $\mathbb{e}'$, $\rho \wr \Gamma \vdash_d \mathbb{e}' : (p_k^s, l_k) \Leftrightarrow \rho \wr \Gamma \vdash_d \oplus(\overline{\mathbb{v}}, \mathbb{e}', \overline{\mathbb{e}}) : \kappa$.
  The $\Leftarrow$ direction follows from (T-Op) combined with (1), (2), (4), and (5).
  The $\Rightarrow$ direction follows from the inversion of (T-Op).
- For $C = \theta(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}})$, by Fig. 4, we have $C[\mathbb{e}] = \theta(\overline{\mathbb{v}}, \mathbb{e}, \overline{\mathbb{e}})$.
  Different sub-cases arise for θ = `filter` | `proj` | `cross` | `agg` (see Fig. 30).
  - For θ = `filter` the only rule that matches $C[\mathbb{e}]$'s shape is (T-Filter). Two sub-cases arise:
    * For $C[\mathbb{e}']$ = `filter`$(\mathbb{e}', \mathbb{e}_\lambda)$, by the inversion of (T-Filter): (1) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : T\{f_i : (p_i^s, l_i)\}_{i \in I}$ (2) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda : \{f_j : (p_j^s, l_j)\}_{j \in J} \rightarrow_{d'} (\mathsf{Bool}, l)$ (3) $J \subseteq I$
      Applying the induction hypothesis to (1) we get some κ' and d', s.t. for all $\mathbb{e}'$ $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}' : \kappa' \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0} C'[\mathbb{e}'] : T\{f_i : (p_i^s, l_i)\}_{i \in I}$.
      It remains to show that for all $\mathbb{e}'$, $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : T\{f_i : (p_i^s, l_i)\}_{i \in I} \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0}$ `filter`$(\mathbb{e}_t, \mathbb{e}_\lambda) : T\{f_i : (p_i^s, l_i \sqcup l)\}$.
      The $\Rightarrow$ direction follows from (T-Filter) combined with (2) and (3).
      The $\Leftarrow$ direction follows from the inversion of (T-Filter).
    * For C = `filter`$(\mathbb{e}_t, \mathbb{e}')$, by the inversion of (T-Filter): (1) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I}$ (2) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : \{f_j : (p_j^s, l_j)\}_{j \in J} \rightarrow_{d'} (\mathsf{Bool}, l)$ (3) $J \subseteq I$
      Applying the induction hypothesis to (2) we get some κ' and d', s.t. for all $\mathbb{e}'$ $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}' : \kappa' \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0} C'[\mathbb{e}'] : \{f_j : (p_j^s, l_j)\}_{j \in J} \rightarrow_{d'} (\mathsf{Bool}, l)$.
      It remains to show that forall $\mathbb{e}'$, $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : \{f_j : (p_j^s, l_j)\}_{j \in J} \rightarrow_{d'} (\mathsf{Bool}, l) \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0}$ `filter`$(\mathbb{e}_t, \mathbb{e}') : T\{f_i : (p_i^s, l_i \sqcup l)\}$.
      The $\Rightarrow$ direction follows from (T-Filter) combined with (1) and (3).
      The $\Leftarrow$ direction follows from the inversion of (T-Filter).
  - For θ = `proj`: similar to `filter`.
  - For θ = `cross`: similar to `filter`.
  - For θ = `agg`: similar to `filter`.
- For $C = \mathsf{encr}(\bullet, s)$, from (56) and Fig. 4, we have $C[\mathbb{e}] = \mathsf{encr}(\mathbb{e}, s)$. (Ext-T-Encr) is the only rule whose conclusion matches $C[\mathbb{e}]$'s shape. From inversion on (Ext-T-Encr), we have (1) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (p^\varnothing, l)$ (2) $\varphi^{\text{ty}}(\mathsf{encr}) = p \longrightarrow p^s$ (3) $(p^s, l) \sqsubseteq d$ (4) $s \neq \varnothing$, and by

induction hypothesis on (1) we get for some κ' and d', s.t. for all $\mathbb{e}'$ $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}' : \kappa' \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0} C'[\mathbb{e}'] : (p^{\varnothing}, l)$
It remains to show that for all $\mathbb{e}'$, $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : (p^{\varnothing}, l) \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0} \mathsf{encr}(\mathbb{e}, s) : (p^s, l)$.
The $\Rightarrow$ direction follows from (Ext-T-Encr) combined with (2), (3) ,and (4).
The $\Leftarrow$ direction follows from the inversion of (Ext-T-Encr).

- For $C = \mathsf{decr}(\bullet)$, from (56) and Fig. 4, we have $C[\mathbb{e}] = \mathsf{decr}(\mathbb{e}')$. Similar to case $C = \mathsf{encr}(\bullet, s)$.
- For $C = \bullet(\overline{\mathbb{e}})$, from (56) and Fig. 4, we have $C[\mathbb{e}_\lambda] = \mathbb{e}_\lambda(\overline{\mathbb{e}})$. (Ext-T-Apply) is the only rule whose conclusion matches $C[\mathbb{e}_\lambda]$'s shape. From inversion on (Ext-T-Apply), we have (1) $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}_\lambda : \overline{\kappa} \rightarrow_{\tilde{d}} \kappa$ (2) $\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : \kappa}$ (3) $\kappa \sqsubseteq d$, and by induction hypothesis on (1) we get for some κ' and d', s.t. for all $\mathbb{e}'$ $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}' : \kappa' \Leftrightarrow \rho \wr \Gamma \Vdash_{d'/d_0} C'[\mathbb{e}'] : \overline{\kappa} \rightarrow_{\tilde{d}} \kappa$
It remains to show that for all $\mathbb{e}'$ $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : \overline{\kappa} \rightarrow_{\tilde{d}} \kappa \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}'(\overline{\mathbb{e}}) : \kappa$.
The $\Rightarrow$ direction follows from (Ext-T-Apply) combined with (2), and (3).
The $\Leftarrow$ direction follows from the inversion of (Ext-T-Apply).
- For $C = \mathbb{v}(\overline{\mathbb{v}}, \bullet, \overline{\mathbb{e}})$, from (56) and Fig. 4, we have $C[\mathbb{e}] = \mathbb{v}(\overline{\mathbb{v}}, \mathbb{e}, \overline{\mathbb{e}})$. (Ext-T-Apply) is the only rule whose conclusion matches $C[\mathbb{e}]$'s shape. From inversion on (Ext-T-Apply), we have $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa_k$ for $k = |\overline{v}| + 1$, and the induction hypothesis applies.
- For $C = \bullet.f$, from (56) and Fig. 4, we have $C[\mathbb{e}] = \mathbb{e}.f$. (Ext-T-RecSelect) is the only rule whose conclusion matches $C[\mathbb{e}]$'s shape. From inversion on (Ext-T-RecSelect), we have $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : \{\overline{f : (p^s, l)}\}$, and the induction hypothesis applies.
- For $C = \{\overline{f : \mathbb{v}}, f : \bullet, \overline{f : \mathbb{e}}\}$, from (56) and Fig. 4, we have $C[\mathbb{e}] = \{\overline{f : v}, f : \mathbb{e}, \overline{f : \mathbb{e}}\}$. (Ext-T-Record) is the only rule whose conclusion matches $C[\mathbb{e}]$'s shape. From inversion on (Ext-T-Record), we have $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : (p^s_k, l_k)$ for $k = |\overline{f : \mathbb{v}}| + 1$, and the induction hypothesis applies.
- For $C = [\bullet]_{d'}$, from (56) and Fig. 4 we have $C[\mathbb{e}] = [\mathbb{e}]_{d'}$. (Ext-T-Return) is the only rule whose conclusion matches $C[\mathbb{e}]$'s shape. From inversion on (Ext-T-Return), we have $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e} : \kappa$, and applying induction hypothesis yields $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e} : \kappa'$ for some κ'.

□

F.3.2 *Substitution lemma*

Lemma 25 (Substitution). *Let $\rho \wr \Gamma \Vdash_{d'} \overline{\mathbb{v} : \kappa}$ and $\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e} : \kappa'$. Then $\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} : \kappa'$.*

Proof. By induction on the derivation of $\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e} : \kappa'$, i.e., rules in Fig. 11, Fig. 29, and Fig. 30. In all cases, $\rho \wr \Gamma \vdash_{d'} \overline{\mathbb{v} : \kappa}$ by assumption.

- Case (Ext-T-Const), $\mathbb{e} = c^s$ and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = c^s$.
By (Ext-T-Const), $\rho \wr \emptyset \Vdash_{d/d_0} c^s : \kappa'$, and the claim follows.
- Case (Ext-T-Op), $\mathbb{e} = \oplus(\overline{\mathbb{e}})$ and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \oplus(\{\overline{\mathbb{v}}/\overline{x}\}\overline{\mathbb{e}})$.
From the conclusion of (Ext-T-Op), $\kappa' = (p^s, \sqcup_i l_i)$.
By the inversion of (Ext-T-Op):
$$\varphi^{\mathrm{ty}}(\oplus) = \overline{p^s} \longrightarrow p^s \quad (57a) \qquad \rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \overline{\mathbb{e} : (p^s, l)} \quad (57b)$$
$$(p^s, \sqcup_i l_i) \sqsubseteq d \quad (57c)$$
Applying induction hypothesis to (57b) we get $\rho \wr \Gamma \Vdash_{d/d_0} \overline{\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} : (p^s, l)}$.
Applying (Ext-T-Op) to the last along with 57a and 57c, it follows that $\rho \wr \Gamma \Vdash_{d/d_0} \oplus(\{\overline{\mathbb{v}}/\overline{x}\}\overline{\mathbb{e}}) : (p^s, \sqcup l_i)$.
- Case (Ext-T-Var), $\mathbb{e} = y$.
By the inversion of (Ext-T-Var): (1) $\kappa' = (\Gamma, \overline{x : \kappa})(y)$ (2) $\kappa' \sqsubseteq d$

  - If $y \notin dom(\overline{x : \kappa})$, then $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = y$.
    It follows that $\Gamma(y) = (\Gamma, \overline{x : \kappa})(y) = \kappa'$, and we get the claim by applying (Ext-T-Var).
  - If $y \in dom(\overline{x : \kappa})$, then $y = x_i$ and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \mathbb{v}_i$ for some i.
    It follows that $\kappa' = (\Gamma, \overline{x : \kappa})(y) = (\Gamma, \overline{x : \kappa})(x_i) = \kappa_i$.
    By assumption we have $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{v}_i : \kappa_i$, by applying Lem. 23 to this assumption and (2), we get $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_i : \kappa_i$.

- Case (Ext-T-Return), $\mathbb{e} = [\mathbb{e}']_{d'}$ and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = [\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}']_{d'}$.
  By the inversion of (Ext-T-Return): $\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d'/d_0} \mathbb{e}' : \kappa'$, applying the induction hypothesis gives (1) $\rho \wr \Gamma \Vdash_{d'/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}' : \kappa'$, and it remains to apply (Ext-T-Return) on (1) to prove what we had set out to prove.
- Case (Ext-T-ConfUp).
  By the inversion of (Ext-T-ConfUp): (1) $\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e} : \kappa$; (2) $\kappa <: \tilde{\kappa}$; and (3) $\tilde{\kappa} \sqsubseteq d$.
  Applying the induction hypothesis to (1) we get $\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} : \kappa$.
  Applying (Ext-T-ConfUp) to the last, (2) and (3), we get $\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} : \tilde{\kappa}$.
- Case (Ext-T-Fun), $\mathbb{e} = \lambda[d^*](\overline{x^* : \kappa^*}).\ \mathbb{e}^*$.
  We can alpha-convert the bound variables $\overline{x^* : \kappa^*}$ in $\lambda[d^*](\overline{x^* : \kappa^*}).\ \mathbb{e}^*$ so that they are different from both: (a) the variables $\overline{x}$ being substituted, and (b) the free variables of $\overline{\mathbb{v}}$. After the conversion it would hold that $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \lambda[d^*](\overline{x^* : \kappa^*}).\ \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}^*$.
  By the inversion of (Ext-T-Fun): (1) $\kappa' = \overline{\kappa^*} \longrightarrow_{d^*} \kappa^*$, and (2) $\rho \wr \Gamma, \overline{x : \kappa}, \overline{x^* : \kappa^*} \Vdash_{d^*/d_0} \mathbb{e}^* : \kappa^*$
  By the *standard permutation lemma* and (2) we have $\rho \wr \Gamma, \overline{x^* : \kappa^*}, \overline{x : \kappa} \Vdash_{d^*/d_0} \mathbb{e}^* : \kappa^*$.
  By the *standard weakening lemma* applied to the assumption $\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v} : \kappa}$ we have $\rho \wr \Gamma, \overline{x^* : \kappa^*} \Vdash_{d/d_0} \overline{\mathbb{v} : \kappa}$.
  Applying the induction hypothesis with $\Gamma = (\Gamma, \overline{x^* : \kappa^*})$ to the last two, we derive

$$\rho \wr \Gamma, \overline{x^* : \kappa^*} \Vdash_{d^*/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}^* : \kappa^*,$$

  and (T-Fun) lets us conclude $\rho \wr \Gamma \Vdash_{d/d_0} \lambda[d^*](\overline{x^* : \kappa^*}).\ \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}^* : \overline{\kappa^*} \longrightarrow_{d^*} \kappa^*$ as needed.
- Case (Ext-T-Apply), $\mathbb{e} = \mathbb{e}_\lambda(\overline{\mathbb{e}})$, and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_\lambda(\overline{\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}})$.
  By the inversion of (Ext-T-Apply):

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}_\lambda : \overline{\kappa^*} \longrightarrow_{d'} \kappa \qquad (58a)$$

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \overline{\mathbb{e} : \kappa^*} \qquad (58b) \qquad\qquad \kappa \sqsubseteq d \qquad (58c)$$

  Applying the induction hypothesis to (58a) and (58b), we get $\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_\lambda : \overline{\kappa^*} \longrightarrow_{d'} \kappa$ and $\rho \wr \Gamma \Vdash_{d/d_0} \overline{\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} : \kappa^*}$.
  Applying (Ext-T-Apply) to the above derivations and (58c) we get the claim.
- Case (Ext-T-Decr), $\mathbb{e} = \mathtt{decr}(\mathbb{e}')$, and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \mathtt{decr}(\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}')$.
  By the inversion of (Ext-T-Decr):

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}' : (p^s, l) \qquad (59a) \qquad\qquad s \neq \varnothing \qquad (59b)$$

$$\varphi^{\mathrm{ty}}(\mathtt{decr}) = p^s \longrightarrow p \qquad (59c) \qquad\qquad (p, l) \sqsubseteq d \qquad (59d)$$

  Applying the induction hypothesis to (59a) we get $\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}' : (p^s, l)$.
  Applying (Ext-T-Decr) to the last, (59c), (59b), and (59d) we get the claim.
- Case (Ext-T-Record), $\mathbb{e} = \{\overline{f : \mathbb{e}}\}$, and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \{\overline{f : \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}}\}$.
  By the inversion of (Ext-T-Record):

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \overline{\mathbb{e} : (p^s, l)} \qquad (60a) \qquad\qquad \forall i.\ (p_i^s, l_i) \sqsubseteq d \qquad (60b)$$

Applying the induction hypothesis to (60a), we get $\rho \wr \Gamma \vdash_d \overline{\{\mathbb{v}/\overline{x}\}\mathbb{e} : (p^s, l)}$.
Applying (Ext-T-Record) to the last and (60b), we get the claim.

- Case (Ext-T-Encr), $\mathbb{e} = \mathsf{encr}(\mathbb{e}', s)$, and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \mathsf{encr}(\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}', s)$.
  By the inversion of (T-Encr):

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}' : (p, l) \qquad (61a) \qquad\qquad s \neq \varnothing \qquad (61b)$$

$$\varphi^{\mathrm{ty}}(\mathsf{encr}) = p \longrightarrow p^s \qquad (61c) \qquad\qquad (p^s, l) \sqsubseteq d \qquad (61d)$$

  Applying the induction hypothesis to (61a) we get $\rho \wr \Gamma \vdash_d \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}' : (p, l)$.
  Applying (Ext-T-Encr) to last, (61c), (61b) and (61d) we get the claim.
- Case (Ext-T-RecSelect), $\mathbb{e} = \mathbb{e}'.f_j$, and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = (\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}').f_j$
  By the inversion of (Ext-T-RecSelect): (1) $\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \{\overline{f : \mathbb{e}'}\} : \{\overline{f : (p^s, l)}\}$.
  Applying the induction hypothesis to (1), we get $\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{f : \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}'}\} : \{\overline{f : (p^s, l)}\}$.
  Applying (Ext-T-RecSelect) to the last, the claim follows.
- Case (Ext-T-Filter), $\mathbb{e} = \mathtt{filter}(\mathbb{e}_t, \mathbb{e}_\lambda)$,
  and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \mathtt{filter}(\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_t, \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_\lambda)$.
  We have $\kappa' = T\{f_i : (p_i^s, l_i \sqcup l)\}$.
  By the inversion of (Ext-T-Filter):

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (62a) \qquad\qquad J \subseteq I \qquad (62b)$$

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}_\lambda : \{f_j : (p_j^s, l_j)\}_{j \in J} \longrightarrow_{d'} (\mathsf{Bool}, l) \quad (62c) \qquad\qquad \forall i.\ (p_i^s, l_i \sqcup l) \sqsubseteq d \qquad (62d)$$

  Applying the induction hypothesis to (62a) and (62c), we get

$$\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \qquad (63a)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_\lambda : \{f_j : (p_j^s, l_j)\}_{j \in J} \longrightarrow_{d'} (\mathsf{Bool}, l) \qquad (63b)$$

  Applying (T-Filter) to the 63a and 63b and also to (62b) and (62d) we get the claim.
- Case (Ext-T-Proj) is similar to (Ext-T-Filter).
- Case (Ext-T-Cross) is similar to (Ext-T-Filter).
- Case (Ext-T-Agg), $\mathbb{e} = \mathtt{agg}(\mathbb{e}_t, f_j, \mathbb{e}_0, \mathbb{e}_\lambda)$, and
  $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \mathtt{agg}(\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_t, f_j, \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_0, \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_\lambda)$.
  We have $\kappa' = T\{\mathsf{key} : (p_j^s, l_j), \mathsf{aggVal} : (p^s, l')\}$.
  By the inversion of (Ext-T-Agg):

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (64a) \qquad\qquad \rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}_0 : (p^s, l') \qquad (64b)$$

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d/d_0} \mathbb{e}_\lambda : (\{f_i : (p_i^s, l_i)\}_{i \in I'}, (p^s, l')) \longrightarrow_{d'} (p^s, l') \quad (64c) \qquad\qquad I' \subseteq I \qquad (64d)$$

$$j \in I \qquad (64e) \qquad\qquad (p^s, l' \sqcup l_j) \sqsubseteq d \qquad (64f)$$

  Applying the induction hypothesis to (64a), (64b), and (64c), we get:

$$\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (65a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_0 : (p^s, l') \quad (65b)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}_\lambda : (\{f_i : (p_i^s, l_i)\}_{i \in I'}, (p^s, l')) \rightarrow_{d'} (p^s, l') \quad (65c)$$

Applying (T-Agg) to (65a),(65b), (65c),(64e), and (64f), we get the claim.

- Case (Ext-T-Bracket), $\mathbb{e} = \langle e_1 \mid e_2 \rangle$, and $\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e} = \langle \{\lfloor \overline{\mathbb{v}} \rfloor_1/\overline{x}\}e_1 \mid \{\lfloor \overline{\mathbb{v}} \rfloor_2/\overline{x}\}e_2 \rangle$
  By the inversion of (Ext-T-Bracket):

$$\rho \wr \Gamma, \overline{x : \kappa} \vdash_d e_1 : \kappa \quad (66a) \qquad \rho \wr \Gamma, \overline{x : \kappa} \vdash_d e_2 : \kappa \quad (66b)$$

$$\kappa \not\sqsubseteq d_0 \quad (66c)$$

  By applying Lem. 22 to the assumption of the lemma, we derive: (i) $\rho \wr \Gamma \vdash_{d'} \overline{\lfloor \mathbb{v} \rfloor_1 : \kappa}$, and (ii) $\rho \wr \Gamma \vdash_{d'} \overline{\lfloor \mathbb{v} \rfloor_2 : \kappa}$.
  Applying the induction hypothesis to (66a) and (i), and also to (66b) and (ii), we get (1) $\rho \wr \Gamma \vdash_d \{\lfloor \overline{\mathbb{v}} \rfloor_1/\overline{x}\}e_1 : \kappa$, and (2) $\rho \wr \Gamma \vdash_d \{\lfloor \overline{\mathbb{v}} \rfloor_2/\overline{x}\}e_2 : \kappa$ Applying (Ext-T-Bracket) to (1), (2), and (66c) we get the claim.
- Case (Ext-T-Bracket-Enc) is similar to (Ext-T-Bracket).
- Case (Ext-T-Bracket-Tbl) is similar to (Ext-T-Bracket).

□

F.3.3 *Proof of subject reduction*

Theorem 8 (Subject Reduction). *Let* $\Vdash_{/d_0} \Omega : \rho$, $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e} : \kappa$ *and* $\mathbb{e} \underset{\Omega}{\Longrightarrow} \mathbb{e}'$ *then* $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{e}' : \kappa$.

Proof. By the induction on the *size of* derivation for evaluations $e \overset{\iota}{\underset{\Omega}{\Longrightarrow}} e'$ for $\iota \in \{\bullet, 1, 2\}$. We proceed by case analysis on the final evaluation rule in the derivation.

- Case (Ext-Ctx)

$$\mathbb{e} = C[\mathbb{e}_1'] \quad (67a) \qquad C[\mathbb{e}_1'] \overset{\iota}{\underset{\Omega}{\Longrightarrow}} C[\mathbb{e}_2'] \quad (67b) \qquad \mathbb{e}_1' \overset{\iota}{\underset{\Omega}{\Longrightarrow}} \mathbb{e}_2' \quad (67c)$$

  By applying Lem. 24 to the premise of the lemma and (67a) we get κ' and d', s.t. for any e':

$$\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}' : \kappa' \Leftrightarrow \rho \wr \Gamma \Vdash_{d/d_0} C[\mathbb{e}'] : \kappa \quad (68a)$$

$$\text{for } e_1', \rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}_1' : \kappa' \quad (68b)$$

  Applying the induction hypothesis to (68b) and (67c) we get $\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{e}_2' : \kappa'$, which we wrap back into the context with (68a).
- Case (Ext-Op)

$$\mathbb{e} = \oplus(\overline{c^s}) \quad (69a) \qquad \overline{c^s} = (c_i^s)_{i \in I} \quad (69b)$$

$$\mathbb{e}' = \mathbb{v}' = c^s = \varphi^{\text{ev}}(\oplus, \overline{c, s}) \quad (69c) \qquad \forall i \in I.\ c_i^s = \lfloor c_i^s \rfloor_1 = \lfloor c_i^s \rfloor_2 \quad (69d)$$

  By the inversion of (Ext-T-Op), which is the only rule matching (69a):

$$\kappa = (p^s, \sqcup l_i) \quad (70a) \qquad \varphi^T = \overline{p^s} \rightarrow p^s \quad (70b)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{e} : (p^s, l)} \quad (70c) \qquad (p^s, \sqcup l_i) \sqsubseteq d \quad (70d)$$

  Due to (69d), only rule (Ext-T-Const) could result in (70c), inverting each we get for every i:

$$\varphi^{\text{ty}}(\mathbb{e}_i) = \varphi^{\text{ty}}(c_i^s) = p_i^s \qquad (71a)$$

Applying (OP-COMP) to (70b), and (71a) for every i, we derive:

$$\varphi^{\text{ty}}(\mathbb{v}') = \varphi^{\text{ty}}(\varphi^{\text{ev}}(\oplus, \overline{c, s})) = p^s \tag{72}$$

Applying (Ext-T-Const) to (72) we conclude $\rho \wr \Gamma \vdash_d \mathbb{v}' : (p^s, \bot)$ and since $\bot \leq \sqcup l_i$, we can use (Ext-T-ConfUp) and (70d) to bump the label and get the claim for $\mathbb{e}'$.

- Case (Ext-Encr)

$$\mathbb{e} = \mathsf{encr}(c^{\varnothing}, s) \qquad (73a)$$

$$\varphi^{\text{ev}}(\mathsf{encr}, c, s) = \mathbb{v}' = \mathbb{e}' \qquad (73b)$$

$$v = \lfloor c^{\varnothing} \rfloor_1 = \lfloor c^{\varnothing} \rfloor_2 \qquad (73c)$$

By the inversion of (Ext-T-Encr), the only rule whose conclusion matches (73a):

$$\kappa = (p^s, l) \quad (74a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} v : (p, l) \quad (74b)$$

$$\varphi^T(\mathsf{encr}) = p \longrightarrow p^s \quad (74c) \qquad (p^s, l) \sqsubseteq d \quad (74d)$$

Due to (73c), only (Ext-T-Const) could result in (74b), inverting which gives:

$$\varphi^{\text{ty}}(v) = p \tag{75}$$

Applying (ENCR-COMP) to (74c) and (75) we get:

$$\varphi^{\text{ty}}(v') = \varphi^{\text{ty}}(\varphi^{\text{ev}}(\mathsf{encr}, v, s)) = p^s \tag{76}$$

Due to monotonicity of $-^c \circ \mathbb{S}$ and (74d) we get $(p^s, \bot) \sqsubseteq d$, which can be used to apply (Ext-T-Const) to (76) and get $\rho \wr \Gamma \vdash_d \mathbb{v}' : (p^s, \bot)$. Since $\bot \leq l$, we can use (Ext-T-ConfUp) and (74d) to bump the label and get the claim for $\mathbb{e}'$.

- Case (Ext-Decr)

$$\mathbb{e} = \mathsf{decr}(c^s) \qquad (77a)$$

$$e' = \varphi^{\text{ev}}(\mathsf{decr}, c, s) = \mathbb{v}' \qquad (77b)$$

$$v = c^s = \lfloor c^s \rfloor_1 = \lfloor c^s \rfloor_2 \qquad (77c)$$

By the inversion of (Ext-T-Decr), the only rule whose conclusion matches (77a):

$$\kappa = (p, l) \qquad (78a)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} v : (p^s, l) \qquad (78b)$$

$$\varphi^T(\mathsf{decr}) = p^s \longrightarrow p \quad (78c) \qquad (p, l) \sqsubseteq d \quad (78d)$$

Due to (77c), only (T-Const) could result in (78b), inverting which gives:

$$\varphi^{\text{ty}}(v) = p^s \tag{79}$$

Applying (DECR-COMP) to (78c) and (79) we get:

$$\varphi^{\text{ty}}(\mathbb{v}') = \varphi^{\text{ty}}(\varphi^{\text{ev}}(\mathsf{decr}, v)) = p \tag{80}$$

Due to $-^c \circ \mathbb{S}$ being order-preserving and (78d) we get $(p^s, \bot) \sqsubseteq d$, which can be used to apply (T-Const) to (80) and get $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}' : (p, \bot)$. Since $\bot \leq l$, we can use (Ext-T-ConfUp) and (78d) to bump the label and get the claim for $\mathbb{e}'$.

- Case (Ext-Tbl)

$$\mathbb{e} = \texttt{table}(name) \quad (81a) \qquad \Omega(name) = \mathbb{v} \quad (81b) \qquad \mathbb{e}' = \mathbb{v} \quad (81c)$$

By the inversion of (T-TblCall), the only rule matching (81a):

$$\kappa = T\{\overline{f : (p^s, l)}\} \quad (82a)$$

$$\rho(name) = T\{\overline{f : (p^s, l)}\} \quad (82b)$$

$$\forall i.\ (p_i^s, l_i) \sqsubseteq d \quad (82c)$$

Applying $\Vdash_{/d_0} \Omega : \rho$ to (82b) and (81b) we get:

$$\mathbb{v} = T\{\overline{f_i : \overline{v_{ij}}^j}^i\} \quad (83a) \qquad \varphi^{\text{ty}}(v_{i,j}) = p_i^s \quad (83b)$$

Applying (Ext-T-Const) to (83b) and (82c), resetting the latter to $\bot$ using monotonicity of $-^c \circ \mathbb{S}$, we get:

$$\rho \wr \Gamma \Vdash_{d/d_0} v_{i,j} : (p_i^s, \bot) \tag{84}$$

After bumping the labels in Proof of subject reduction back up using (Ext-T-ConfUp) and (82c), we can, finally, apply (Ext-T-Tbl) to get the claim.

- Case (Ext-TblProj) similar to (Ext-Tbl), except for Lem. 22 applied at the end.
- Case (Ext-OpQuery), $\mathbb{e}$ is $\theta(\overline{\mathbb{v}})$.

$$\mathbb{e} = \theta(\overline{\mathbb{v}}) \quad (85a) \qquad \mathbb{v}_k \neq \langle v_k^1 \mid v_k^2 \rangle \quad (85b)$$

$$\varphi^O(\theta, \lfloor \overline{\mathbb{v}} \rfloor_\iota) = v'_\iota \quad (85c) \qquad \mathbb{e}' = v'_1 \star v'_2 \quad (85d)$$

– $\theta = \texttt{filter}$, $\mathbb{e} = \texttt{filter}(\mathbb{v}_t, \mathbb{v}_\lambda)$. By the inversion of (Ext-T-Filter) matching (85a)

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (86a) \qquad J \subseteq I \quad (86b)$$

$$\kappa = T\{f_i : (p_i^s, l_i \sqcup l)\} \quad (86c)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_\lambda : \{f_j : (p_j^s, l_j)\}_{j \in J} \rightarrow_{d'} (\texttt{Bool}, l) \quad (86d)$$

$$\forall i.\ (b_i, l_i \sqcup l) \sqsubseteq d \quad (86e)$$

By the inversion of (Ext-T-Tbl), the only rule matching (86a) and (85b):

$$\mathbb{v}_t = T\{\overline{\overline{f_i : \mathbb{v}_{i,j}}^i}^j\} \quad (87a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v}_{i,j}}^i : (p_i^s, l_i) \quad (87b)$$

Applying (Ext-T-Record) to (87b) using only indices $J \subset I$:

$$\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{f_i : \mathbb{v}_{i,j}}^{i \in J}\} : \{\overline{f_i : (p_i^s, l_i)}^{i \in J}\} \tag{88}$$

Applying (Ext-T-Apply) to (88), (86d), and (86e) stepped down using monotonicity of $-^c \circ \mathbb{S}$, we get:

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_\lambda(\{\overline{f_j : \mathbb{v}_{j,k}}^{j\in J}\}) : (\texttt{Bool}, l) \tag{89}$$

By the inversion of (PrimEv Filter) applied to (85c):

$$\lfloor \mathbb{v}_\lambda \rfloor_\iota(\{\overline{f_j : \lfloor \mathbb{v}_{j,k} \rfloor_\iota}^{j\in J}\}) \xrightarrow[\Omega]{}^* v_k^\iota \tag{90a}$$

$$v_k^\iota \in \{\texttt{true}, \texttt{false}\} \tag{90b}$$

$$K^\iota_{\texttt{true}} = \{k \in K : v_k^\iota = \texttt{true}\} \tag{90c}$$

$$\varphi^{\text{ev}}(\texttt{filter}, T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_\iota}^{i\in I}}^{k\in K}\}, \lfloor \mathbb{v}_\lambda \rfloor_\iota) = T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_\iota}^{i\in I}}^{k\in K^\iota_{\texttt{true}}}\} \tag{91}$$

From Th. 7 applied to (90a):
$$\mathbb{v}_\lambda(\{\overline{f_j : \mathbb{v}_{j,k}}^{j\in J}\}) \xrightarrow[\Omega]{}^* \mathbb{v}_k$$

Now, by induction on the number of reduction steps for (92a) and by using (89):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_k : (\texttt{Bool}, l) \tag{93}$$

There are two cases: $\mathbb{v}_k \neq \langle v_k^1 \mid v_k^2 \rangle$ for all k or there exists some $\tilde{k}$, s.t. $\mathbb{v}_{\tilde{k}} = \langle v_{\tilde{k}}^1 \mid v_{\tilde{k}}^2 \rangle$. In the former case we have $K^1_{\texttt{true}} = K^2_{\texttt{true}} = K_{\texttt{true}}$:

$$T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_1}^{i\in I}}^{k\in K_{\texttt{true}}}\} \star T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_2}^{i\in I}}^{k\in K_{\texttt{true}}}\} = T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_1 \star \lfloor \mathbb{v}_{i,k} \rfloor_2}^{i\in I}}^{k\in K_{\texttt{true}}}\} \tag{94}$$

Using Projection and encoding cancel to get $\lfloor \mathbb{v}_{i,k} \rfloor_1 \star \lfloor \mathbb{v}_{i,k} \rfloor_2 = \mathbb{v}_{i,k}$, we are done with this case.
In the latter case, we have $\mathbb{v}_{k^*} = \langle v_{k^*}^1 \mid v_{k^*}^2 \rangle$, by the inversion of (Ext-T-Bracket), the only matching rule, we know that $(d_0, \varnothing) \notin \mathbb{S}(l)$. Since $l \preccurlyeq \sqcap_i(l_i \sqcup l)$ we can use monotonicity of $-^c \circ \mathbb{S}$ to derive $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_i(l_i \sqcup l))$. Now we can use (Ext-T-Bracket-Tbl) to type $v_1' \star v_2'$.

– $\theta = \texttt{proj}$, $\mathbb{e} = \texttt{proj}(\mathbb{v}_t, \mathbb{v}_\lambda)$. By the inversion of (Ext-T-Proj), the only rule matching (85a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_t : T\{f_i : (p_i^s, l_i)\}_{i\in I} \tag{95a}$$

$$I' \subseteq I \tag{95b}$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_\lambda : \{f_i : (p_i^s, l_i)\}_{i\in I'} \rightarrow_{d'} \{f_j : (p_j^s, l_j)\}_{j\in J} \tag{95c}$$

$$\forall j \in J.\ (p_j^s, l_j \sqcup (\sqcap_{i\in I} l_i)) \sqsubseteq d \tag{95d}$$

By the inversion of (Ext-T-Tbl), the only rule matching (95a) and (85b):

$$\mathbb{v}_t = T\{\overline{\overline{f_i : \mathbb{v}_{i,k}}^{i}}^{k}\} \tag{96a}$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v}_{i,k}}^{i} : (p_i^s, l_k) \tag{96b}$$

Applying (Ext-T-Record) to (96b) using only indices $I' \subseteq I$:

$$\rho \wr \Gamma \Vdash_{d/d_0} \{\overline{f_i : \mathbb{v}_{i,k}}^{i \in I'}\} : \{\overline{f_i : (p_i^s, l_i)}^{i \in I'}\} \tag{97}$$

Applying (Ext-T-Apply) to (97), (95c), and (95d) stepped down using monotonicity of $-^c \circ \mathbb{S}$, we get:

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_\lambda(\{\overline{f_i : \mathbb{v}_{i,k}}^{i \in I'}\}) : \{\overline{f_i : (p_i^s, l_i)}^{i \in J}\} \tag{98}$$

By the inversion of (PrimEv Proj) applied to (85c):

$$\forall k \in K. \lfloor \mathbb{v}_\lambda \rfloor_\iota (\{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_\iota}^{i \in I}\}) \xrightarrow[\Omega]{}^* \{\overline{f_i : v_{i,k}^\iota}^{i \in J}\} \tag{99a}$$

$$\varphi^{\text{ev}}(\texttt{proj}, T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_\iota}^{i \in I}}^{k \in K}\}, \lfloor \mathbb{v}_\lambda \rfloor_\iota) = T\{\overline{\overline{f_i : v_{i,k}^\iota}^{i \in J}}^{k \in K}\} \tag{99b}$$

From Completeness of extended language evaluation applied to (99a):

$$\mathbb{v}_\lambda(\{\overline{f_j : \mathbb{v}_{j,k}}^{j \in J}\}) \xrightarrow[\Omega]{}^* \mathbb{v}'_k \tag{100}$$

Now, by induction on the number of reduction steps for (100) and by using (98):

$$\rho \wr \Gamma \vdash_d \mathbb{v}'_k : \{\overline{f_i : (p_i^s, l_i)}^{i \in J}\} \tag{101a}$$

$$\lfloor \mathbb{v}'_k \rfloor_\iota = \{\overline{f_i : v_{i,k}^\iota}^{i \in J}\} \tag{101b}$$

By the inversion of (Ext-T-Record), the only rule matching (101a):

$$\mathbb{v}'_k = \overline{f_i : \mathbb{v}'_{i,k}}^{i \in J} \tag{102a}$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v}'_{i,k}}^{i \in J} : (p_i^s, l_k) \tag{102b}$$

Since $\mathbb{v}'_{i,k} = \lfloor \mathbb{v}'_{i,k} \rfloor_1 \star \lfloor \mathbb{v}'_{i,k} \rfloor_2$ by Lem. 18 from (101b):

$$\begin{aligned} T\{\overline{\overline{f_i : v_{i,k}^1}^{i \in J}}^{k \in K}\} \star T\{\overline{\overline{f_i : v_{i,k}^2}^{i \in J}}^{k \in K}\} \\ = T\{\overline{\overline{f_i : \mathbb{v}'_{i,k}}^{i \in J}}^{k \in K}\} \end{aligned} \tag{103}$$

Applying (T-Tbl) to (102b), we get the claim.

- $\theta = \texttt{cross}$, $\mathbb{e} = \texttt{cross}(\mathbb{v}_1, \mathbb{v}_2)$. By the inversion of (Ext-T-Cross), the only rule matching (85a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_1 : T\{f_i : (p_i^s, l_i)\}_{i \in I_1} \tag{104a}$$

$$I_1 \cap I_2 = \emptyset \tag{104b}$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_2 : T\{f_i : (p_i^s, l_i)\}_{i \in I_2} \tag{104c}$$

By the inversion of (Ext-T-Tbl), the only rule matching (104a), (104c), and (85b):

$$\mathbb{v}_1 = T\{\overline{\overline{f_i : \mathbb{v}_{i,k}}^{i \in I_1}}^{k \in K_1}\} \tag{105a}$$

$$\mathbb{v}_2 = T\{\overline{\overline{f_i : \mathbb{v}_{i,k}}^{i \in I_2}}^{k \in K_2}\} \tag{105b}$$

$$\forall k \in K_1. \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v}_{i,k}}^{i \in I_1} : (p_i^s, l_k) \tag{105c}$$

$$\forall k \in K_2. \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v}_{i,k}}^{i \in I_2} : (p_i^s, l_k) \tag{105d}$$

By the inversion of (PrimEv Join) applied to (85c):

$$\begin{aligned} \varphi^O(\texttt{cross}, \lfloor \mathbb{v}_1 \rfloor_\iota, \lfloor \mathbb{v}_2 \rfloor_\iota) = \\ T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k_1} \rfloor_\iota}^{i \in I_1}, \overline{f_i : \lfloor \mathbb{v}_{i,k_2} \rfloor_\iota}^{i \in I_2}}^{k_1,k_2 \in K_1 \times K_2}\} \end{aligned} \tag{106}$$

It remains to apply (Ext-T-Tbl) to (105c) and (105d) to get the claim.

– $\theta = \mathsf{agg}$, $\mathbb{e} = \mathsf{agg}(\mathbb{v}_t, f_j, \mathbb{v}_0, \mathbb{v}_\lambda)$. By inverting (T-Agg), the only rule matching (85a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (107a) \qquad (p^s, l' \sqcup l_j) \sqsubseteq d \quad (107b)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_0 : (p^s, l') \quad (107c) \qquad j \in I \quad (107d)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_\lambda : (\{f_i : (p_i^s, l_i)\}_{i \in I}, (p^s, l')) \longrightarrow_{d'} (p^s, l') \quad (107e)$$

By the inversion of (Ext-T-Tbl), the only rule matching (107a) and (85b):

$$\mathbb{v}_t = T\{\overline{\overline{f_i : \mathbb{v}_{i,k}}^{i}}^{k}\} \quad (108a) \qquad \forall k, \forall i. \rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_{i,k} : (p_i^s, l_k) \quad (108b)$$

By the inversion of (PrimEv Agg) applied to (85c):

$$\begin{aligned} C^\iota &= \{\lfloor \mathbb{v}_{i,k} \rfloor_\iota : k \in K\};\ \{k^\iota_{c,1}, \dots, k^\iota_{c,m^\iota_c}\} \\ &= \{k : \lfloor \mathbb{v}_{j,k} \rfloor_\iota = c\};\ v^\iota_{c,0} = \lfloor \mathbb{v}_0 \rfloor_\iota \end{aligned} \quad (109)$$

$$\lfloor \mathbb{v}_\lambda \rfloor_\iota(\{\overline{f_i : \lfloor \mathbb{v}_{i,k^\iota_{c,s}} \rfloor_\iota}^{i}\}, v^\iota_{c,s-1}) \xrightarrow[\Omega]{}^* v^\iota_{c,s} \quad (110)$$

$$\begin{aligned} &\varphi^{\text{ev}}(\mathsf{agg}, T\{\overline{\overline{f_i : \lfloor \mathbb{v}_{i,k} \rfloor_\iota}^{i}}^{k \in K}\}, f_j, \lfloor \mathbb{v}_0 \rfloor_\iota, \lfloor \mathbb{v}_\lambda \rfloor_\iota) \\ &\quad = T\{\overline{\mathtt{key} : c, \mathtt{aggVal} : v^\iota_{c,m^\iota}}^{c \in C^\iota}\} \end{aligned} \quad (111)$$

There are two main cases: either for every k, $\mathbb{v}_{k,j} \neq \langle \lfloor \mathbb{v}_{k,j} \rfloor_1 \mid \lfloor \mathbb{v}_{k,j} \rfloor_2 \rangle$ or there exists some $\tilde{k}$, s.t. $\mathbb{v}_{\tilde{k},j} = \langle \lfloor \mathbb{v}_{\tilde{k},j} \rfloor_1 \mid \lfloor \mathbb{v}_{\tilde{k},j} \rfloor_2 \rangle$. In the latter case, by the inversion of (Ext-T-Bracket) the only rule matching (108b) for $\tilde{k}$ we know that $(d_0, \emptyset) \notin \mathbb{S}(l_j)$. Since $l_j = l_j \sqcap (l_j \sqcup l')$, we can use (Ext-T-Bracket-Tbl) to type $v'_1 \star v'_2$.
Now, the former case: for every k, $\mathbb{v}_{k,j} \neq \langle \lfloor \mathbb{v}_{k,j} \rfloor_1 \mid \lfloor \mathbb{v}_{k,j} \rfloor_2 \rangle$, which due (107d) and (108b) and inversion of (Ext-T-Const) means $\mathbb{v}_{k,j} = \lfloor \mathbb{v}_{k,j} \rfloor_\iota$, $C^1 = C^2 = C$, $m^1_c = m^2_c = m_c$, and $k^1_{c,s} = k^2_{c,s} = k_{c,s}$, which due to Completeness of extended language evaluation, implies that $v^\iota_{c,m^\iota_c} = \lfloor \mathbb{v}'_{c,m_c} \rfloor_\iota$, where $\mathbb{v}'_{c,m_c}$ is defined recursively with $\mathbb{v}'_{c,0} = \mathbb{v}_0$ and $\mathbb{v}_\lambda(\{\overline{f_i : \mathbb{v}_{i,k_{c,s}}}^{i}\}, \mathbb{v}'_{c,s-1}) \xrightarrow[\Omega]{}^* \mathbb{v}'_{c,s}$. Applying the induction hypothesis on the number of derivation steps for (110) and (T-Apply) we have $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}'_{c,s} : (p^s, l')$, so that (Ext-T-ConfUp) can bump it to $\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}'_{c,s} : (p^s, l' \sqcup l_j)$, and we can use (Ext-T-Tbl) to type the result.

- Case (Ext-Apply)

$$\mathbb{e} = \lambda[d'](\overline{x : \kappa}).\mathbb{e}''(\overline{\mathbb{v}}) \quad (112a) \qquad \mathbb{e}' = [\{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}'']_d \quad (112b)$$

By the inversion of (Ext-T-Apply), the only rule whose conclusion matches (112a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \lambda[d'](\overline{x : \kappa}).\mathbb{e}'' : \overline{\kappa} \longrightarrow_{d'} \kappa \quad (113a)$$

$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v} : \kappa} \quad (113b) \qquad \kappa \sqsubseteq d \quad (113c)$$

By the inversion of (Ext-T-Fun), the only rule whose conclusion matches (113a):

$$\rho \wr \Gamma, \overline{x : \kappa} \Vdash_{d'/d_0} \mathbb{e}'' : \kappa \quad (114)$$

Applying Substitution to and (114) we further get:

$$\rho \wr \Gamma \Vdash_{d'/d_0} \{\overline{\mathbb{v}}/\overline{x}\}\mathbb{e}'' : \kappa \quad (115)$$

Applying (Ext-T-Return) to (115) and (113c) we get the claim.

- Case (Ext-Return)

$$\mathbb{e} = [\mathbb{v}]_{d'} \quad (116a) \qquad \mathbb{e}' = \mathbb{v} \quad (116b)$$

By the inversion of (Ext-T-Return), which is the only rule that matches (116a):

$$\rho \wr \Gamma \Vdash_{d'/d_0} \mathbb{v} : \kappa \quad (117a) \qquad \kappa \sqsubseteq d \quad (117b)$$

By applying Lem. 23 to (117a) and (117b) we get the claim.

- Case (Ext-RecSelect)

$$\mathbb{e} = \{\overline{f : \mathbb{v}}\}.f_k \quad (118a) \qquad \mathbb{e}' = \mathbb{v}_k \quad (118b)$$

By the inversion of (Ext-T-RecSelect), the only rule that matches (118a):

$$\kappa = (p^s_k, l_k) \quad (119a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \{\overline{f : \mathbb{v}}\} : \{\overline{f : (p^s, l)}\} \quad (119b)$$

By the inversion of (Ext-T-Record), the only rule that matches (119b), the claim follows.

- Case (Ext-LiftOp)

$$\mathbb{e} = \oplus(\overline{\mathbb{v}}) \quad (120a) \qquad \mathbb{v}_k = \langle v^1_k \mid v^2_k \rangle \quad (120b) \qquad \mathbb{e}' = \langle \oplus(\lfloor \overline{\mathbb{v}} \rfloor_1) \mid \oplus(\lfloor \overline{\mathbb{v}} \rfloor_2) \rangle \quad (120c)$$

By the inversion of (Ext-T-Op), the only rule that matches (120a):

$$\kappa = (p^s, \sqcup_i l_i) \quad (121a) \qquad \varphi^{\text{ty}}(\oplus) = \overline{p^s} \longrightarrow p^s \quad (121b) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v} : (p^s, l)} \quad (121c)$$

$$p^s_i = p^{s'}_i \Rightarrow p^s = p^{s'} \quad (121d) \qquad (p^s, \sqcup_i l_i) \sqsubseteq d \quad (121e) \qquad s, \overline{s} \in \{s', \varnothing\} \quad (121f)$$

Applying Lem. 22 to (121c) we get:

$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\lfloor \mathbb{v} \rfloor_1 : (p^s, l)} \quad (122a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \overline{\lfloor \mathbb{v} \rfloor_2 : (p^s, l)} \quad (122b)$$

Applying (T-Op) to (121b), (121f),(121e), and both (122a) and (122b):

$$\rho \wr \Gamma \Vdash_{d/d_0} \oplus(\lfloor \overline{\mathbb{v}} \rfloor_1) : (p^s, l) \quad (123a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \oplus(\lfloor \overline{\mathbb{v}} \rfloor_2) : (p^s, l) \quad (123b)$$

The case analysis on (121c) applied to $\mathbb{v}_k$ from (120b):

- By the inversion of (Ext-T-Bracket): $(p^s_k, l_k) \not\sqsubseteq d_0$, which, expanding the definition of $\sqsubseteq$ implies either of the two:
  - $p^s = (p^s_k, l_k)$ and $(d_0, s) \notin \mathbb{S}(l_k)$. The former together with Equation 121f implies $p^s = p^s_k$ for some p, while the latter combined with monotonicity of $-^c \circ \mathbb{S}$ and $l_k \preccurlyeq \sqcup_i l_i$ gives $(d_0, s) \notin \mathbb{S}(\sqcup_i l_i)$, which implies:
  $$p^s = p^s_k \wedge (p^s, \sqcup_i l_i) \not\sqsubseteq d_0 \quad (124)$$
  - $p^s = (p_k, l_k)$ and $(d_0, \varnothing) \notin \mathbb{S}(l_k)$. The latter implies $(d_0, \varnothing, \sqcup_i l_i) \notin d_0$ by and $l_k \preccurlyeq \sqcup_i l_i$, doing case analysis on p^s we get:
  $$p^s = p^s \wedge (p, \sqcup_i l_i) \not\sqsubseteq d_0 \quad (125a)$$
  $$p^s = p \wedge (p, \sqcup_i l_i) \not\sqsubseteq d_0 \quad (125b)$$
- By the inversion of (Ext-T-Bracket-Enc):
$$p^s = (p^s_k, l_k) \quad (126a) \qquad (d_0, \varnothing) \notin \mathbb{S}(l) \quad (126b)$$
By (126a) and (121f) we conclude that $p^s = p^s_k$
, while applying monotonicity of $-^c \circ \mathbb{S}$ to (126b) and $l_k \preccurlyeq \sqcup_i l_i$ we get $(d_0, \varnothing) \notin \mathbb{S}(\sqcup_i l_i)$. Combining we get
$$p^s = p^s \wedge (p, \sqcup_i l_i) \not\sqsubseteq d_0 \quad (127)$$

To get the claim we either apply to (122a) and (122b) (Ext-T-Bracket), if we have (125b) or (124), or (Ext-T-Bracket-Enc), if we have (127) or (125a).

- Case (Ext-LiftOpQuery)

$$\mathbb{e} = \theta(\overline{\mathbb{v}}) \quad (128a) \qquad \mathbb{v}_k = \langle v_k^1 \mid v_k^2 \rangle \quad (128b) \qquad \mathbb{e}' = \langle \theta(\lfloor \overline{\mathbb{v}} \rfloor_1) \mid \theta(\lfloor \overline{\mathbb{v}} \rfloor_2) \rangle \quad (128c)$$

– Case $\theta = \texttt{proj}$, $\mathbb{e} = \texttt{proj}(\mathbb{v}_t, \mathbb{v}_\lambda)$.
By the inversion of (Ext-T-Proj), the only rule matching (128a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (129a)$$
$$\forall j \in J.\ (p_j^s, l_j \sqcup (\sqcap_{i \in I} l_i)) \sqsubseteq d \quad (129b)$$
$$I' \subseteq I \quad (129c)$$
$$\kappa = T\{f_j : (p_j^s, l_j \sqcup (\sqcap l_i))\}_{j \in J} \quad (129d)$$
$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_\lambda : \{f_i : (p_i^s, l_i)\}_{i \in I'} \rightarrow_{d'} \{f_j : (p_j^s, l_j)\}_{j \in J} \quad (129e)$$

We can only have (128b) for $\mathbb{v}_t$, since there is no typing rule for $\mathbb{v}_\lambda$, inverting the only matching (Ext-T-Bracket-Tbl): $(d_0, \varnothing) \notin \mathbb{S}(\sqcap l_i)$. since $\sqcap l_j \subseteq \sqcup(\sqcap l_i)$, we can use monotonicity of $-^c \circ \mathbb{S}$ and (Ext-T-Bracket-Tbl) to get the claim.

– $\theta = \texttt{cross}$, $\mathbb{e} = \texttt{cross}(\mathbb{v}_1, \mathbb{v}_2)$. By the inversion of (Ext-T-Cross), the only rule matching (85a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_1 : T\{f_i : (p_i^s, l_i)\}_{i \in I_1} \quad (130a)$$
$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_2 : T\{f_i : (p_i^s, l_i)\}_{i \in I_2} \quad (130b)$$
$$\kappa = T\{f_k : (p_k^s, l_k \sqcup (\sqcap_{i \in I} l_i) \sqcup (\sqcap_{i \in J} l_i))\}_{k \in I \cup J} \quad (130c)$$

Without loss of generality, let us assume that (128b) holds for (130a) then by inversion of the only matching rule (Ext-T-Bracket-Tbl), $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_{i \in I_1} l_i)$. Since

$$\sqcap_{i \in I} l_i \leq \sqcap_{k \in I \cup J}(l_k \sqcup (\sqcap_{i \in I} l_i) \sqcup (\sqcap_{i \in J} l_i))$$

by monotonicity of $-^c \circ \mathbb{S}$ we also have $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_{k \in I \cup J}(l_k \sqcup (\sqcap_{i \in I} l_i) \sqcup (\sqcap_{i \in J} l_i)))$, and we can use (Ext-T-Bracket-Tbl) to get the claim.

– $\theta = \texttt{filter}$, $\mathbb{e} = \texttt{filter}(\mathbb{v}_t, \mathbb{v}_\lambda)$. By the inversion of (Ext-T-Filter) the only rule matching (128a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (131a) \qquad \kappa = T\{f_i : (p_i^s, l_i \sqcup l)\} \quad (131b)$$

It can only be (128b) for $\mathbb{v}_t$, and by the inversion of (Ext-T-Bracket-Tbl), $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_{i \in I} l_i)$ since $\sqcap_{i \in I} l_i \preccurlyeq \sqcap_{i \in I}(l_i \sqcup l)$, using monotonicity of $-^c \circ \mathbb{S}$ we get $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_{i \in I} l \sqcup l_i)$, and it remains to apply (Ext-T-Bracket-Tbl) to get the claim.

– $\theta = \texttt{agg}$, $\mathbb{e} = \texttt{agg}(\mathbb{v}_t, f_j, \mathbb{v}_0, \mathbb{v}_\lambda)$. By inverting (Ext-T-Agg), the only rule matching (128a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_t : T\{f_i : (p_i^s, l_i)\}_{i \in I} \quad (132a)$$
$$\rho \wr \Gamma \Vdash_{d/d_0} \mathbb{v}_0 : (p^s, l') \quad (132b)$$
$$j \in I \quad (132c)$$

Assume first that (128b) holds for $\mathbb{v}_t$, then we can invert (Ext-T-Bracket-Tbl) as the only rule matching (132a) and get $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_{i \in I} l_i)$. Naturally, $\sqcap_{i \in I} l_i \preccurlyeq l_j$, hence by monotonicity of $-^c \circ \mathbb{S}$ we get $(d_0, \varnothing) \notin \mathbb{S}(l_j)$, and we can use (Ext-T-Bracket-Tbl) to get the claim.

- Case (Ext-LiftEncr)

$$\mathbb{e} = \mathsf{encr}(\langle v_1 \mid v_2 \rangle, s) \quad (133a) \qquad \mathbb{e}' = \langle \mathsf{encr}(v_1, s) \mid \mathsf{encr}(v_2, s) \rangle \quad (133b)$$
$$\kappa = (p^s, l) \quad (133c)$$

By the inversion of (Ext-T-Encr), the only rule that matches (133a):

$$s \neq \varnothing \quad (134a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \langle v_1 \mid v_2 \rangle : (p, l) \quad (134b)$$

$$\varphi^{\mathrm{ty}}(\mathsf{encr}) = (p, s) \to p^s \quad (134c) \qquad (p^s, l) \sqsubseteq d \quad (134d)$$

By the inversion of (Ext-T-Bracket), the only rule that matches (134b):

$$\rho \wr \Gamma \vdash_d v_i : (p, l) \quad (135a) \qquad (p, l) \not\sqsubseteq d_0, \quad (135b)$$

By applying (T-Encr) to (135a), (134a), (134c), and (134d):

$$\rho \wr \Gamma \vdash_d \mathsf{encr}(v_i, s) : (p^s, l) \quad (136)$$

By applying (Ext-T-Bracket) to (136) and (135b) we get the claim.

- Case (Ext-LiftDecr)

$$\mathbb{e} = \mathsf{decr}(\langle v_1 \mid v_2 \rangle) \quad (137a) \qquad \mathbb{e}' = \langle \mathsf{decr}(v_1) \mid \mathsf{decr}(v_2) \rangle \quad (137b)$$

$$\kappa = (p, l) \quad (137c)$$

By the inversion of (Ext-T-Decr), the only rule that matches (137a):

$$s \neq \varnothing \quad (138a) \qquad \rho \wr \Gamma \Vdash_{d/d_0} \langle v_1 \mid v_2 \rangle : (p^s, l) \quad (138b)$$

$$\varphi^{\mathrm{ty}}(\mathsf{decr}) = (p^s) \to p \quad (138c) \qquad (p, l) \sqsubseteq d \quad (138d)$$

There are sub-cases: either (138b) is derived using (Ext-T-Bracket) or (Ext-T-Bracket-Enc)

– By the inversion of (Ext-T-Bracket):

$$\rho \wr \Gamma \vdash_d v_i : (p^s, l) \quad (139a) \qquad (p^s, l) \not\sqsubseteq d_0 \quad (139b)$$

By applying using the fact that $-^c \circ \mathbb{S}$ maps to $\mathcal{A}$ and that adversaries are downwards closed w.r.t. $\preccurlyeq_{ds}$ to (139b) we get:

$$(p, l) \not\sqsubseteq d_0 \quad (140)$$

– By the inversion of (Ext-T-Bracket-Enc):

$$\rho \wr \Gamma \vdash_d v_i : (p^s, l) \quad (141a) \qquad (p, l) \not\sqsubseteq d_0 \quad (141b)$$

By applying (T-Decr) to (138c), (138a), (138d), and either (139a) or (141a), we get:

$$\rho \wr \Gamma \vdash_d \mathsf{decr}(v_i) : (p, l) \quad (142)$$

Applying (Ext-T-Bracket) to (142) and either (140) or (141b) we get the claim.

- Case (Ext-LiftRecSelect)

$$\mathbb{e} = \langle \{\overline{f : v}\} \mid \{\overline{f : w}\} \rangle.f_k \quad (143a) \qquad \mathbb{e}' = \langle \{\overline{f : v}\}.f_k \mid \{\overline{f : w}\}.f_k \rangle \quad (143b)$$

$$\kappa = (p_k^s, l_k) \quad (143c)$$

By the inversion of (Ext-T-RecSelect), the only rule that matches (143a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \{f : \langle \{\overline{f : v}\} \mid \{\overline{f : w}\} \rangle\} : \{f : \{\overline{f : (p^s, l)}\}\} \quad (144a)$$

No rule's conclusion matches (144a).

- Case (Ext-LiftApply)

$$\mathbb{e} = \langle v_1 \mid v_2 \rangle(\overline{\mathbb{v}}) \quad (145a) \qquad \mathbb{e}' = \langle v_1 \lfloor \overline{\mathbb{v}} \rfloor_1 \mid v_2 \lfloor \overline{\mathbb{v}} \rfloor_2 \rangle \quad (145b)$$

By the inversion of (Ext-T-Apply), the only rule that matches (145a):

$$\rho \wr \Gamma \Vdash_{d/d_0} \langle v_1 \mid v_2 \rangle : \overline{\kappa} \rightarrow_{\tilde{d}} \kappa \quad (146a)$$
$$\rho \wr \Gamma \Vdash_{d/d_0} \overline{\mathbb{v} : \kappa} \quad (146b)$$
$$\kappa \sqsubseteq d \quad (146c)$$

There is no rule, whose conclusion matches (146a).

- Case (Ext-Bracket) w.l.o.g. $\iota = 1$ and $\zeta = 2$

$$\mathbb{e} = \langle e_1 \mid e_2 \rangle \quad (147a) \qquad e_1 \overset{1}{\underset{\Omega}{\Longrightarrow}} e_1' \quad (147b) \qquad e' = \langle e_1' \mid e_2 \rangle \quad (147c)$$

By the inversion of either (Ext-T-Bracket), or (Ext-T-Bracket-Enc), or (Ext-T-Bracket-Tbl), which are the only rules whose conclusion matches (147a):

$$\rho \wr \Gamma \vdash_d e_1 : \kappa \quad (148a) \qquad \rho \wr \Gamma \vdash_d e_2 : \kappa \quad (148b) \qquad \text{side condition on } \kappa \quad (148c)$$

Applying the induction hypothesis to (148a) and (147b) we get:

$$\rho \wr \Gamma \vdash_d e_1' : \kappa \quad (149)$$

Applying either (Ext-T-Bracket), or (Ext-T-Bracket-Enc), or (Ext-T-Bracket-Tbl) to (149), (148b), and (148c) we get the claim.

□

F.4 Soundness

F.4.1 Inaccessible case

Lemma 26 (Inaccessible values are related). *If $(d, \varnothing) \in \mathbb{S}(l)$, then for any non-arrow type κ and non-function values v_1, v_2, s.t., $\rho \vdash_d v_1 : \kappa$ and $\rho \vdash_d v_2 : \kappa$ it must hold that $v_1 \sim^{d,l} v_2$.*

Proof. Induction over κ, where due to $-^c \circ \mathbb{S}$ mapping to $\mathscr{A}$ and the elements of the latter being downwards closed w.r.t. $\preccurlyeq_{ds}$ the premise of (EquivConst$^{\text{OUT}}$) holds vacuously. □

Proof of Lem. 1. Follows from Th. 4 and Lem. 26. □

F.4.2 Encoding and decoding are correct

Proof of Lem. 4. We use induction over derivation of $\rho \Vdash_d \mathbb{v} : \kappa$ discarding cases that are impossible for values:

- Cases (T-Tbl) and (T-Record): by the induction hypothesis we know that all the entries are pair-wise related, hence we can apply (EquivTblPW$^{\text{OUT}}$) and (EquivRec$^{\text{OUT}}$), respectively.
- Case (T-Const): we can directly apply either (EquivEq$^{\text{OUT}}$) or (EquivEncEq$^{\text{OUT}}$).
- Case (Ext-T-Bracket): $v = \langle c_1^s \mid c_2^s \rangle$, $(p^s, l) \not\sqsubseteq d$, but due to the branches being typeable in the same domain we must have $(p^s, l) \sqsubseteq d$, a contradiction.
- Case (Ext-T-Bracket-Enc): $v = \langle c_1^s \mid c_2^s \rangle$, $(d, \varnothing, l) \notin \mathbb{S}^{/l}$, the latter implies $l \preccurlyeq l$. Due to branches being typeable in d, we have $(d, s, l) \in \mathbb{S}^{/l}$, and, hence $(d, s) \in \mathbb{S}(l)$. Using monotonicity of $-^c \circ \mathbb{S}$ we derive $(d, s) \in \mathbb{S}(l)$, and we can apply (EquivConst$^{\text{OUT}}$).
- Case (Ext-T-Bracket-Tbl): $(d_0, \varnothing) \notin \mathbb{S}(\sqcap_i l_i)$, which implies $l \preccurlyeq l_i$ for any i. Now we can apply the reasoning similar to (Ext-T-Bracket-Enc) and relate all the entries by (EquivConst$^{\text{OUT}}$) then finally applying (EquivTblAll$^{\text{OUT}}$).

□

Proof of Lem. 2. We consider each name n in order and then perform induction over derivation of $v_1 \sim^l_{\rho(n)} v_2$, where $v_1 = \Omega_1(n)$ and $v_2 = \Omega_2(n)$.

- Cases (EquivEq$^{\text{IN}}$) and (EquivEncEq$^{\text{IN}}$) are trivial since there are no brackets involved.
- Cases (EquivRec$^{\text{IN}}$), and (EquivTblPW$^{\text{IN}}$) follow easily from the induction hypothesis and either (T-Tbl) or (T-Record).
- Case (EquivConst$^{\text{IN}}$). We know from the premise of the rule that $\kappa = (p^s, l)$ and from the premise of the theorem that $(d_0, \varnothing) \notin \mathbb{S}(l)$, so we can conclude that $(p^{\varnothing}, l) \not\sqsubseteq d_0$ and apply either (Ext-T-Bracket-Enc) or (Ext-T-Bracket).

□

F.4.3 *Non-interference*

Lemma 27 (Compatibility is monotone w.r.t. policy). *If for all l $\mathbb{S}(l) \subseteq \mathbb{S}'(l)$, then $\kappa \sqsubseteq d$ w.r.t. $\mathbb{S}$ implies $\kappa \sqsubseteq d$ w.r.t. $\mathbb{S}'$.*

Proof. Straightforward from the definition of $\kappa \sqsubseteq d$. □

Lemma 28 (Typing is monotone w.r.t. policy). *If for all l $\mathbb{S}(l) \subseteq \mathbb{S}'(l)$, then $\rho \vdash_d e : \kappa$ w.r.t. $\mathbb{S}$ implies $\rho \vdash_d e : \kappa$ w.r.t. $\mathbb{S}'$.*

Proof. Induction over derivation of $\rho \vdash_d e : \kappa$ applying Lem. 27 where needed. □

Proof of Lem. 3. It is easy to see that for all l $\mathbb{S}(l) \subseteq \mathbb{S}^{/l}(l)$, so the claim follows from Lem. 28.

□

Proof of Th. 1. Due to Lem. 1 we assume $(d, \varnothing) \notin \mathbb{S}(l)$. Consider an expression e for which know $\rho \vdash_d e : \kappa$ w.r.t. a given $\mathbb{S}$. Now let us consider any level l and any two stores Ω_1 and Ω_2 satisfying ρ, s.t., $\Omega_1 \sim_{\rho}^{d,l} \Omega_2$ w.r.t $\mathbb{S}$ and two values u_1 and u_2, $e_1 \underset{\Omega}{\longrightarrow}^{*} u_1$ and $e \underset{\Omega}{\longrightarrow}^{*} u_2$. Let $\mathbb{\Omega} = \Omega_1 \star \Omega_2$, by Lem. 17 we have $\lfloor \mathbb{\Omega} \rfloor_i = \Omega_i$, $i \in \{1, 2\}$. By Lem. 2 from $\Omega_1 \sim_{\rho}^{d,\perp} \Omega_2$ we derive that $\Vdash_{/d} \Omega_1 \star \Omega_2 : \rho$. Also, it is easy to see that for all l $\mathbb{S}(l) \subseteq \mathbb{S}^{/l}(l)$, hence $\rho \vdash_d e : \kappa$ holds w.r.t. $\mathbb{S}^{/l}$. As the derivation rules for $\Vdash_d$ are a superset of those for $\vdash_d$, we have $\rho \Vdash_d e : \kappa$ and by subject reduction Th. 8 we have $\rho \Vdash_d \mathbb{v} : \kappa$. Hence, we can apply Th. 7 to $e \underset{\Omega}{\longrightarrow}^{*} u_1$ and $e \underset{\Omega}{\longrightarrow}^{*} u_2$, which gives us $\mathbb{v}$ such that $e \underset{\mathbb{\Omega}}{\Longrightarrow}^{*} \mathbb{v}$ and $\lfloor \mathbb{v} \rfloor_i = u_i$, $i \in \{1, 2\}$. Finally, it remains to apply Lem. 4 to derive that $u_1 = \lfloor \mathbb{v} \rfloor_1 \sim_{\kappa}^{d,\perp} \lfloor \mathbb{v} \rfloor_2 = u_2$. Since we chose Ω_1, Ω_2, u_1, u_2, and l arbitrarily, we have shown $\mathbb{S}\text{-NI}(e)_{\rho,d}$. □

Received 2022-11-10; accepted 2023-03-31