A Type-based Dimensional Analysis for XQuery *

Zhendong Su
<su@cs.ucdavis.edu>

Gary Wassermann
<wassermg@cs.ucdavis.edu>

Department Computer Science
University of California, Davis
Davis, CA 95616-8562 USA

ABSTRACT

XML transformation languages, such as XQuery, take an XML document as input and produce another XML document as output. It is useful to know *statically* that such transformations always produce valid documents, for static debugging of the transformation program or for eliminating dynamic checks on the output documents. This gives rise to the *XML type checking problem*. Type- and automata-theoretic techniques have been proposed to address this type checking problem, exploiting XML's tree structure. However, existing approaches are not capable of reasoning about *dimensional* information of produced XML documents, such as that two locations in the output documents always have the same number of elements.

This paper presents a type-based analysis for XQuery to discover dimensional relationships in output documents from XQuery programs through refined type checking. For example, it can identify program fragments producing the same number of elements for all input documents. The novel aspects of our analysis are techniques to deal with the rich tree structure of XML types, whereas array analyses (*e.g.*, bounds checking) for languages such as C deal with flat arrays. In this paper, we present our type system and give a sketch of its soundness.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Type Structure*

General Terms

Languages

Keywords

XML Type Checking, XQuery, Tree Types, Size Analysis, Dimensional Relationships

*This research was supported in part by a faculty research grant to Zhendong Su from the University of California, Davis.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

1. INTRODUCTION

Since XML [2] became a W3C recommendation in 1998, XML has been increasingly accepted as the standard format for electronic data exchange. XML has also gained acceptance as a format for data storage because it generalizes relational databases.

Because two parties who wish to exchange data generally organize their data differently, one or both of the parties must transform their data so that it is suitable for the other to use. In the context of XML, "schemas" [18] specify data organization. When data is exchanged using XML, the recipient specifies a schema to which all received XML documents must conform. The sender has the task of writing a transformation program to convert data from his own schema to the recipient's schema. If the sender can determine that his program performs the transformation correctly, no runtime checks are necessary.

This gives rise to the XML type checking problem. Let T be the set of all XML documents (T is a mnemonic for "trees"; all XML documents have a tree structure). An XML type is a subset of all documents: $\tau \subseteq T$, often called a *schema*. The XML type checking problem asks, for source and target types τ_s and τ_t respectively, and transformation program P, is it true that $\forall x \in \tau_s.P(x) \in \tau_t$ [21]? One common approach to answer this question is based on type inference: an output type τ' is conservatively inferred based on the program and the source type: $P(\tau_s) \subseteq \tau'$. If the inferred type is a subtype of the target type, $\tau' \subseteq \tau_t$, then the program successfully type checks.

We introduce the notion of dimensions in XML documents and types: a *dimension* denotes the number of XML elements and/or scalars in a consecutive sequence under a common parent.[1] For a particular XML document, dimensions are always known constants. However, dimensions may not remain constant across all documents in a single type. In this case, the dimensions of the type are represented by variables, which may be constrained to allow only values valid for some document within the type. When some dimensions of a type are constrained in terms of other dimensions, we call those *dimensional relations*. Because of the use of the Kleene star as a type constructor, we do not seek to discover the values of dimensions; in general that is impossible. Rather, we seek to discover relationships among dimensions.

Some practical settings require dimensional information. For example, in a document that has parallel lists of movie titles and the years those movies were made, the length of those lists can vary provided that they are equal to each other. As another example, consider a specification manual that must include the same information in multiple languages. The number of headings in one linguistic section can vary provided that it equals the number of head-

[1]The term "dimension" alludes to vectors, where a vector of n elements is an n-dimensional vector.

```
1 doc[
2   titles[
3           /catalog/book/title ],
4   isbns[
5           /catalog/book/isbn ]
6 ]
```

Figure 1: Given that each book has exactly one title and one ISBN, the number of titles in the listing must equal the number of ISBNs.

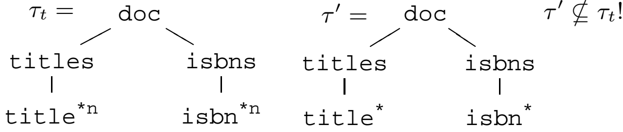

Figure 2: A target type with dimensional relations. Conservatively inferred types using existing techniques cause correct programs to fail to type check.

ings in every other linguistic section. Dimensional relationships arise in settings that include parallel or repeated data.

No previous technique for XML type checking can accurately type check a program when the target type has dimensional relations and the program output is not confined to a regular subtype of the output type. In [17], the decidability of XML type checking is established using k-pebble tree transducers when no dimensional relations are present. It is unclear how well these automata-based techniques would work in practice because of their high computational complexity. Existing type-based approaches [3] provide more practical solutions, but currently are unable to infer types with dimensional relations. In this paper, we present a type-based approach to reason about dimensional relations for transformation programs written in XQuery.

Several transformation languages have been proposed for XML transformations including XSLT [7], XQuery [3], HaXml [23], and Relaxer [9]. Among these, XQuery stands out as a subject for analysis for two reasons. First, it will soon be a W3C recommendation, which means that it will be widely used and even more widely supported. Second, it has a static type system intended to address the XML type checking problem.

Consider the XQuery program (with modified syntax) shown in Figure 1. This program takes as input an XML catalog of books and creates a listing of titles and ISBNs, perhaps for easy ISBN lookup by title in a printed listing. Because each book has exactly one title and one ISBN, the lists rooted at `titles` and `isbns` must have the same number of elements—that is, they must have the same dimension. Because a correspondence is expected between elements in the lists, the programmer would like to confirm that this dimensional relationship holds.

Figure 2 gives the output type τ_t (omitting the scalar children of `title` and `isbn`) as the programmer intends it. Using existing type inference methods, however, the type τ' would be inferred. Because $\tau' \not\subseteq \tau_t$, this correct program fails to type check. Given the current practice in data transformation, this situation is relatively common.

1.1 Difficulties with Dimensional Inference

At first consideration, it may seem as though inferring dimensional relationships could be accomplished using integer constraints as done for array analyses in languages such as C, but surprisingly

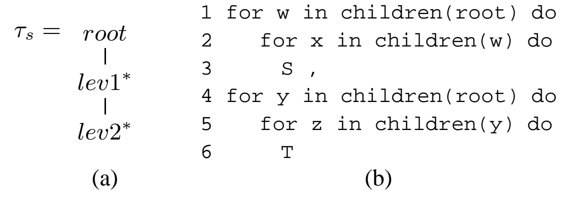

Figure 3: A source type with nested repetitions.

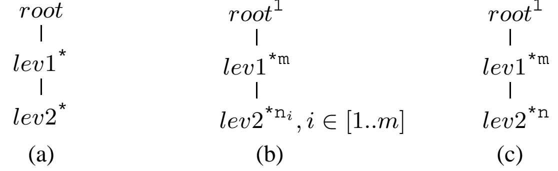

Figure 4: Two ways to annotate a source type τ_s.

it is not that simple. The main problem is that for XML transformations, we need to deal with a much richer data structure and interrelate tree sub-structures. On the other hand, standard array analyses apply to flat structures, namely arrays, and how size information about these arrays flows in a program.

Consider the source type τ_s and program shown in Figure 3. The semantics of paths in XQuery programs (as on lines `3` and `5` in Figure 1) can be achieved with nested `for` expressions alternating with `case` expressions.[2] Lines `1`–`3` and `4`–`6` of the program in Figure 3 have the same semantics as $/root/\tilde{}/\tilde{}$ (where ˜ is a wildcard that matches any tag), except line `3` substitutes an `S` for whatever the output would have been, and equivalently with `T` on line `6`. If the alternating `case` expressions had been included, the two halves of the program would have had the semantics of $/root/lev1/lev2$, excepting the substitutions.

Clearly this program produces the same number of `S`'s as `T`'s. However, standard type systems do a modular analysis, using only the types of subexpressions and the global environment to infer types. Therefore in discovering a relation, such as dimensional equality, between two expressions, the type system is restricted to using information it has available at the time of typing both expressions: the global environment, *i.e.*, the input type. It must discover relations between the expressions and the input type to relate the expressions. Suppose that in hoping to discover precise dimensional relationships of each list, we annotate τ_s as in Figure 4b, where `1`, `m`, and $\mathtt{n}_i$ are dimension annotations for the corresponding types.

We argue that the precision aimed at cannot be achieved. The `for` expression has the form (`for` x `in` e_1 `do` e_2). The expression e_1 evaluates to a list, and for each element `a` in that list, x gets bound to `a` and e_2 gets executed. There are two approaches to typing the `for` expression. We first explore the approach used in XQuery's type system [3]. Under this approach, x is bound to the union of the unit types in e_1's type, τ_1, and e_2 is typed once with x having that binding. Type constructors are then added to the inferred type based on their occurrences in τ_1.

In inferring a type for the program in Figure 3b, if τ_s is annotated as shown in Figure 4b, the type of expression `children(root)` on line `1` is τ_1, as shown in Figure 5. Figure 5 also shows the rest of the types we refer to in this paragraph. Suppose we find the type

[2] For example, in Figure 14, lines `1`–`10` match the semantics of `/catalog/book/title`.

$$\tau_1 = lev1^{*\mathtt{m}} \quad / \quad lev2^{*\mathtt{n}_i}, i \in [1..m] \qquad \vdots \qquad \tau_u = lev1^1 \quad / \quad lev2^{*\mathtt{n}_1|\cdots|\mathtt{n}_m}$$

$$\tau_{1b} = lev2^{*\mathtt{n}_1|\cdots|\mathtt{n}_m}$$
$$\Rightarrow \tau_{1b} = lev2^{*\mathtt{r}}, \{\mathtt{min(n}_i\mathtt{)} \leq \mathtt{r} \leq \mathtt{max(n}_i\mathtt{)}\}$$

$$\tau_{ub} = lev2^1 \qquad \vdots \qquad \tau_{2b} = S^1$$

$$\tau' = S^{*\mathtt{p}}, \{\mathtt{min(m}\times\mathtt{n}_i\mathtt{)} \leq \mathtt{p} \leq \mathtt{max(m}\times\mathtt{n}_i\mathtt{)}\}$$

Figure 5: Some inferred types in our first attempt to infer all dimensional relationships precisely.

$$\tau_{u_1} = lev1^1 \quad / \quad lev2^{*\mathtt{n}_1} \qquad \tau_{u_2} = lev1^1 \quad / \quad lev2^{*\mathtt{n}_2} \qquad \tau_{u_3} = lev1^1 \quad / \quad lev2^{*\mathtt{n}_3} \qquad \cdots$$

Figure 6: Some inferred types in our second attempt to infer all dimensional relationships precisely.

we will assign to w within the body of the for expression by taking the union of the unit types in τ_1, as done in XQuery's type system. This yields τ_u. The type of children(w) on line 2 is then τ_{1b}, however, it is not clear what this dimension annotation means. To add clarity, we rewrite it as shown in Figure 5. We can now find the unit type to which x gets assigned as τ_{ub}, and consequently the body of the inner for expression on line 3 is typed as τ_{2b}. We then go back up and compose the type τ_{2b} with *r. When going up again to find the type of the for expression on line 1, we compose the type of the nested for expression with *m. The result is τ' (for clarity, the constraints have been simplified, so r is not shown).

Unfortunately, all we know about p's value is that it is confined to a given range. When the for expression on line 4 is typed, the result will also be a starred type whose dimension is confined to the same range. Knowing that two numbers are in the same range is usually not sufficient to relate the two numbers in any concrete way, such as describing one as a function of the other. Consequently, knowing that two dimensions are in the same range would not help in any of the examples discussed earlier, so we are not interested in constraints that simply confine dimensions to ranges.

The second approach to typing the for expression is the one taken in [8]: find a type for the body of a for expression for every named unit type in τ_1 and combine them based on the type structure of τ_1. Given the annotation for τ_s in Figure 4b, the type of children(root) is again τ_1, as shown in Figure 5. Because the number of children ($\mathtt{n}_i$) may be different for each of the m $lev1$'s, the first unique unit type here is τ_{u_1}, as shown in Figure 6. The second is τ_{u_2}, the third is τ_{u_3}, and so on. Trying to infer a type for the for expression by inferring a type for e_2 based on τ_{u_1}, τ_{u_2}, τ_{u_3}, ..., is cumbersome and requires complicated symbolic reasoning about summations such as $\sum_{i=1}^{m} \mathtt{n}_i$.

Why cannot we get precise dimension information through precise source type annotations? When a type element in a tree type has a Kleene star (*e.g.*, $lev1^*$ in Figure 3), its children in the type tree (*e.g.*, $lev2^*$) represent uniformly all lists of child elements of the starred element in an actual document. Adding precise dimension annotations to Kleene starred elements (*e.g.*, $lev2^{*\mathtt{n}_i}$) of the input type distinguishes within the type tree the concrete lists that the Kleene starred element represents. After distinctions have been added to the input type, either the type system "factors out" the distinctions, resulting in a loss of precision (as shown in Figure 5), or in addressing the distinctions directly, the type system faces increased complexity (as shown in Figure 6). In this paper, we present an approach to overcome problems with these two attempted methods. Next we briefly discuss our approach.

1.2 Our Approach

In XQuery, there is no mechanism to access the elements in a list non-uniformly. For example, it is impossible to remove the first element from a given list. We take advantage of this in our analysis. The key insight of our approach is that the elements of a list are treated uniformly, both in the type and in the program, so the only information we need is the *total* number of elements in a concrete tree represented by an element (or more precisely, by a path) in the type tree. We annotate the source type as shown in Figure 4c. The annotation n on $lev2$ in Figure 4c denotes the total number of $lev2$ elements in the input document that have as parent a $lev1$ element and as grandparent a $root$ element. Note the difference between this annotation and a dimension: the elements may not all have a common parent. For an alternating sequence of for and case expressions that have the same semantics as the path $/root/lev1/lev2$, the body of the innermost case will get executed n times. Consequently our type system multiplies the inferred dimension of the body of the innermost case expression by n to find the dimension of the outermost for expression.

Because our annotations do not introduce distinctions into τ_s, we avoid the trouble shown in Figure 6. We therefore leverage the more powerful second approach to typing the for expression in our type system. The union operation used in the first approach, which XQuery's type system uses, loses information whenever an element type has more than one child type and so cannot achieve the precision necessary to infer dimensional relationships.

Conditional expressions are often used to select certain elements of a list and pass over others, so they influence the dimensions of output types. A solution that leads to dimensions confined to ranges has the same problems as discussed in Section 1.1, but unless all parts of the boolean expression are static, we cannot determine statically which branch of the conditional will be executed. To address this we use *pair* types. Like conditional types [1], pair types preserve the relationship between the types of the branches of conditional expressions and true and false evaluations of the boolean expression. We relate the dimension of a pair type to the dimensions of the conditional expression's true and false branches as well as to the identity of the boolean condition. If, in a post-processing phase, two boolean conditions can be found to be equivalent, then it becomes possible to relate the dimensions of the corresponding conditional expressions.

1.3 The Organization of This Paper

The rest of this paper is organized as follows: We describe formally in Section 2.1 the subset of XQuery we use, and give its operational semantics in Section 2.2. In Sections 3.1 and 3.2, we present our type system, discussing its type language and type rules. We state a soundness theorem for our type system in Section 3.3. Finally, Section 4 surveys related work, and Section 5 concludes.

2. THE SOURCE LANGUAGE

In this section, we formally define the subset of XQuery that we consider by giving its syntax and operational semantics.

2.1 Syntax

Figure 7 gives the syntax of the subset of XQuery that we consider in this paper. In practice, XQuery programs are evaluated

$$\frac{}{E \vdash c_{\text{int}} \Downarrow c_{\text{int}}}[\textsc{Int}] \quad \frac{E(x) = v}{E \vdash x \Downarrow v}[\textsc{Var}] \quad \frac{a \in \texttt{String} \quad E \vdash e \Downarrow v}{E \vdash a[e] \Downarrow a[v]}[\textsc{Elt}] \quad \frac{E \vdash e_1 \Downarrow v_1 \quad E \vdash e_2 \Downarrow v_2}{E \vdash e_1, e_2 \Downarrow v_1, v_2}[\textsc{Seq}]$$

$$\frac{\begin{array}{l} v_1, v_2 \in \texttt{Integer} \\ E \vdash e_1 \Downarrow v_1 \quad E \vdash e_2 \Downarrow v_2 \end{array}}{E \vdash e_1 + e_2 \Downarrow v_1 + v_2}[\textsc{Plus}] \quad \frac{E \vdash e_1 \Downarrow u_1 \quad E \vdash e_2 \Downarrow u_2}{E \vdash e_1 = e_2 \Downarrow u_1 = u_2}[\textsc{Eq}] \quad \frac{E \vdash e_1 \Downarrow v_1 \quad E \uplus \{x \mapsto v_1\} \vdash e_2 \Downarrow v_2}{E \vdash \texttt{let } x = e_1 \texttt{ do } e_2 \Downarrow v_2}[\textsc{Let}]$$

$$\frac{E \vdash e_b \Downarrow \texttt{true} \quad E \vdash e_1 \Downarrow v_1}{E \vdash \texttt{if } e_b \texttt{ then } e_1 \texttt{ else } e_2 \Downarrow v_1}[\textsc{Cond1}] \quad \frac{E \vdash e_b \Downarrow \texttt{false} \quad E \vdash e_2 \Downarrow v_2}{E \vdash \texttt{if } e_b \texttt{ then } e_1 \texttt{ else } e_2 \Downarrow v_2}[\textsc{Cond2}]$$

$$\frac{E \vdash e_0 \Downarrow u \quad u \in \text{Dom}(p) \quad E \uplus \{x \mapsto u\} \vdash e_1 \Downarrow v_1}{E \vdash \texttt{case } e_0 \texttt{ of } x \texttt{ : } p \texttt{ => } e_1 \texttt{ | } x_2 \texttt{ => } e_2 \Downarrow v_1}[\textsc{Case1}] \quad \frac{E \vdash e_0 \Downarrow u \quad u \notin \text{Dom}(p) \quad E \uplus \{x_2 \mapsto u\} \vdash e_2 \Downarrow v_2}{E \vdash \texttt{case } e_0 \texttt{ of } x \texttt{ : } p \texttt{ => } e_1 \texttt{ | } x_2 \texttt{ => } e_2 \Downarrow v_2}[\textsc{Case2}]$$

$$\frac{E \vdash e \Downarrow a[v]}{E \vdash \texttt{children(}e\texttt{)} \Downarrow v}[\textsc{Child}] \quad \frac{\begin{array}{c} E \vdash e_1 \Downarrow u\texttt{,}v \\ E \uplus \{x \mapsto u\} \vdash e_2 \Downarrow v_u \\ E \vdash \texttt{for } x \texttt{ in } v \texttt{ do } e_2 \Downarrow v' \end{array}}{E \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow v_u\texttt{,}v'}[\textsc{ForElt}] \quad \frac{E \vdash e_1 \Downarrow \texttt{()}}{E \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow \texttt{()}}[\textsc{ForNil}]$$

Figure 8: Operational semantics.

tag	a		
variable	x		
integer	c_{int}	$::= \cdots \mid -1 \mid 0 \mid 1 \mid \cdots$	
string	c_{str}	`::= ""	"a"	"b"	···	"aa"	···`	
boolean	c_{bool}	`::= false	true`	
constant	n	$::= c_{\text{int}} \mid c_{\text{str}} \mid c_{\text{bool}}$	
operator	*op*	`::= +	-	and	or	=	<	···`	
expression	e	::= n	scalar constant
		| x	variable
		| `a[e]`	element
		| `e , e`	sequence
		| `()`	empty sequence
		| `if e then e else e`	conditional
		| `let x = e do e`	local binding
		| `for x in e do e`	iteration
		| `case e of x:p => e`	
		| `x => e end`	case
		| e *op* e	plus, equal, ...
		| `children(e)`	children
pattern	p	$::= a$	element
		| `~`	wildcard
		| s	scalar
data	d	::= n	scalar constant
		| `a[d]`	element
		| `d , d`	sequence
		| `()`	empty sequence

Figure 7: Subset of XQuery that we consider.

by first translating them into a fully expressive subset of the language known as the XQuery Core. The language we consider here follows the pattern of XQuery Core with a few omissions for a simpler presentation. Most conspicuously absent is the path expression (shown in Figure 1 on lines 3 and 5). As mentioned in Section 1.1, path expressions need not be included because their semantics can be achieved with nested `for` and `case` expressions. Beyond that, our subset of XQuery does not include, for example, sorting, explicit type casts, functions, and modules. We do not expect much difficulty in extending our technique to cover these language constructs.

Note that the `case` expression matches a value against a p, defined by the "pattern" derivation. Also, as shown by the "data" derivation, we denote XML elements as `tag[...]` rather than `<tag>...</tag>` to simplify notation.

2.2 Dynamic Semantics

An expression e evaluates according to the operational semantics given in Figure 8. Syntactically, the environment E is defined as:

$$E ::= \emptyset \mid E \uplus \{x \mapsto v\}$$

and E maps a variable, x, to a value, v:

$$E : Var \rightarrow Val$$

The mapping operates according to the following rules:

$$\begin{array}{rcll} E \uplus \{x \mapsto v\}(x') & = & v & \text{if } x = x' \\ & = & E(x') & \text{otherwise} \end{array}$$

Values are defined as data in Figure 7. $E \vdash e \Downarrow v$ is read: expression e evaluates to value v when the variables in e have the values defined in environment E.

The first two rows of Figure 8 are straightforward. The data model given in Figure 7 shows scalars, elements, and sequences as being data in addition to being valid XQuery expressions. Unit values (*i.e.*, elements and scalars) are specified by u as opposed to v, as in [Eq]. The [Let] rule assigns to x in e_2 the value to which e_1 evaluates. In the next row, rules [Cond1] and [Cond2] show that the branch of the conditional expression that gets evaluated and returned depends on the value of the boolean expression. The [Case] rules make use of a Dom() function, which tests whether or not u is in the domain of the pattern p. The value u is in the domain of p if:

- p is a tag, and u is an element whose tag is p; or
- p is a wildcard tag (~), and u is an element; or
- p is a scalar type, and u is a scalar of type p.

Otherwise, u is not in the domain of p. On the bottom row, [Child] is the dual of [Elt]. Finally, the last two rules show the evaluation of the `for` expression. In the first rule, u is the first unit value in the list. The expression e_2 is evaluated once for each unit value produced by e_1 with x bound to each unit value successively.

3. A TYPE SYSTEM FOR DIMENSIONAL ANALYSIS

tag	a	
dim type	$r ::= \mathtt{c}$	constant
	$\mid \mathtt{n}$	variable
scalar type	$s ::= \mathtt{String}$	
	$\mid \mathtt{Boolean}$	
	$\mid \mathtt{Integer}$	
unit type	$u ::= a[\tau]$	element
	$\mid \tilde{}[\tau]$	wildcard
	$\mid s$	scalar type
type	$z ::= \tau\ ,\ \tau$	sequence
	$\mid \tau\ \mid\ \tau$	choice
	$\mid <\tau\ ,\ \tau>$	pair
	$\mid \tau*$	repetition
	$\mid ()$	empty sequence
	$\mid \emptyset$	empty choice
annotated type	$\tau ::= u^1$	unit size
	$\mid z^r$	type and size

Figure 9: Type language.

constraint	$\mathtt{C} ::= \mathtt{C} \cup \{\mathtt{n}\ = e\}$	constraint on variable
	$\mid \mathtt{C} \cup \{\pi\}$	path
	$\mid \emptyset$	empty set
expression	$e ::= \mathtt{c}$	constant
	$\mid \mathtt{n}$	variable
	$\mid (\Gamma_\pi(\pi))$	maps π to annotation
	$\mid (e)$	parentheses
	$\mid e + e$	plus
	$\mid e \times e$	times
	$\mid e\ /\ e$	divide
path	$\pi ::= \pi / a$	tag
	$\mid \varepsilon$	empty tag

Figure 10: Constraint language.

3.1 Type Language

Figure 9 gives our type language. The "dim type" shows that either a constant or a variable can be used as a dimensional annotation to a type. The dimensional annotation denotes the number of unit values that may be matched to the annotated type. The wildcard unit type, $\tilde{}[\tau]$, introduced in [8], is defined such that $\mathtt{a}[\tau]$ is a subtype of $\tilde{}[\tau]$ for all tags $\mathtt{a}$.

Among the types z, "sequence" is the type of two values in sequence. "Choice" is a union type; a value whose type is either of the choices may match it. We introduce the "pair" type for the purpose of typing conditional statements. Like the choice type, a value of either of the two types may match it. Unlike the choice type, the order of the two types is preserved, *i.e.*, $(\tau_1\ \mid\ \tau_2) = (\tau_2\ \mid\ \tau_1)$, but $<\tau_1, \tau_2> \neq <\tau_2, \tau_1>$. Because the order is preserved, it is possible to reason about the types of different conditional expressions in relation to each other.

Figure 10 shows our constraint language. The purpose of the constraints is to allow for reasoning about the dimensions of types in relation to each other. The first kind of constraints therefore assigns to a dimension variable equality to an arithmetic expression. Unique to these expressions is the mapping using the path environment Γ_π. A *valid path* is a path where the first tag matches some root-level tag(s) in τ_s by being identical to it (them) or is ~, in which case it matches all root-level tags, and each successive tag matches in the same way some element(s) in τ_s which are children of elements matched by the tag's predecessor. For example, $/root/lev1/lev2$ and $/root/\tilde{}/lev2$ are both valid paths for τ_s as given in Figure 4. However, if $lev1$ in τ_s were replaced with ~, then $/root/lev1/lev2$ would not be a valid path even though in a valid input tree such a sequence may occur. The path is invalid because our annotations of τ_s tell us nothing about how many $lev2$'s in a $/root/lev1/lev2$ path there are. If the path π being mapped by Γ_π is a valid path in the source type τ_s, $\Gamma_\pi(\pi)$ returns a sum of constants and variables from the annotated τ_s representing the number of unit values this path includes. If π is not a valid path in τ_s, $\Gamma_\pi(\pi)$ returns a fresh variable. This is equivalent to returning "unknown" because the variable will not appear in any other constraints.

Constraints may also include a path which is not explicitly related to anything else in the constraints or types. Such a path remains in the constraint set while the nested `for` and `case` expressions that match the semantics of the path as an expression are being typed. In the last phase of typing that sequence of expressions, the path will be extracted from the constraint set and mapped with the Γ_π environment. The way that the path and Γ_π get connected in our type inference algorithm is explained through an example in Section 3.2.3.

3.2 Type Rules

We use a constraint-based formulation of our type rules. The type judgment $\Gamma \vdash e : \tau, \mathtt{C}$ is read: in environment Γ, expression e has type τ, where the dimension variables in Γ and τ are subject to the constraints $\mathtt{C}$. Type environments are defined by the following grammar:

$$\Gamma ::= \emptyset \mid \Gamma \uplus \{x : \tau\} \mid \Gamma \uplus \{\mathtt{for}\ x : \tau\}$$

where $\{\mathtt{for}\ x : \tau\}$ is used in typing the `for` expression. A type environment, Γ, maps variables and "`for`" variables to types:

$$\Gamma : Var \rightarrow Type$$

The mapping operates according to the following rules:

$$\begin{array}{lcll} \Gamma \uplus \{x : \tau\}(x') & = & \tau & \text{if } x = x' \\ & = & \Gamma(x') & \text{otherwise} \\ \Gamma \uplus \{\mathtt{for}\, x : \tau\}(x') & = & \tau & \text{if } x = x' \\ & = & \Gamma(x') & \text{otherwise} \end{array}$$

The complete list of type rules is given in Figures 11 and 12. In these rules, z, u, and τ are as in Figure 9: z is a type without a dimensional annotation, u is a unit type without an annotation, and τ is a type with an annotation. The particulars of the rules in Figure 12 will be discussed in Section 3.2.3.

We explain here the typing of the four most interesting expressions to dimension types in increasing order of difficulty. In Section 3.2.1, we explain the type rule for sequence expressions, in which two subexpressions are put in sequence. In Section 3.2.2, we explain the type rule plus pre- and post-processing for conditionals. In Section 3.2.3, we explain the typing of the `for` expression, which is the most involved because it is the expression responsible for producing subtrees of unknown size. In Section 3.2.4, we explain the `case` expression, which is often used with `for` and can be typed more precisely than normal conditional expressions.

3.2.1 Sequence Expressions

The type rule for sequence expressions is as follows:

$$\frac{\begin{array}{cc} \tau_1 = z_1^{\mathtt{m}_1} & \tau_2 = z_2^{\mathtt{m}_2} \\ \Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 & \Gamma \vdash e_2 : \tau_2,\ \mathtt{C}_2 \end{array}}{\Gamma \vdash e_1, e_2 : (\tau_1, \tau_2)^{\mathtt{n}},\ \mathtt{C}_1\ \cup\ \mathtt{C}_2\ \cup\ \{\mathtt{n} = \mathtt{m}_1 + \mathtt{m}_2\}}$$

$$\frac{}{\Gamma \vdash () : ()^0,\ \{\}}\,[\textsc{Nil}] \qquad \frac{}{\Gamma \vdash c_{\text{int}} : \mathtt{Int}^1,\ \{\}}\,[\textsc{Int}] \qquad \frac{(x:\tau) \in \Gamma}{\Gamma \vdash x : \tau,\ \{\}}\,[\textsc{Var}]$$

$$\frac{\Gamma \vdash e : a[\tau]^1,\ \mathtt{C}}{\Gamma \vdash \mathtt{children}(e) : \tau,\ \mathtt{C}}\,[\textsc{Child}] \qquad \frac{\Gamma \vdash e : \tau,\ \mathtt{C}}{\Gamma \vdash a[e] : a[\tau]^1,\ \mathtt{C}}\,[\textsc{Elt}]$$

$$\frac{\Gamma \vdash e_1 : \mathtt{Int}^1,\ \mathtt{C}_1 \quad \Gamma \vdash e_2 : \mathtt{Int}^1,\ \mathtt{C}_2}{\Gamma \vdash e_1 + e_2 : \mathtt{Int}^1}\,[\textsc{Plus}] \qquad \frac{\Gamma \vdash e_1 : u^1,\ \mathtt{C}_1 \quad \Gamma \vdash e_2 : u^1,\ \mathtt{C}_2}{\Gamma \vdash e_1 = e_2 : \mathtt{Bool}^1}\,[\textsc{Eq}]$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \quad \Gamma \vdash e_2 : \tau_2,\ \mathtt{C}_2 \\ \tau_1 = z^{\mathtt{m}_1} \qquad \tau_2 = z^{\mathtt{m}_2}\end{array}}{\begin{array}{c}\Gamma \vdash e_1, e_2 : (\tau_1, \tau_2)^{\mathtt{n}}, \\ \mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = \mathtt{m}_1 + \mathtt{m}_2\}\end{array}}\,[\textsc{Seq}]$$

$$\frac{\begin{array}{c}\Gamma \vdash e_b : \mathtt{Boolean}^1,\ \mathtt{C}_b \\ \Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \quad \Gamma \vdash e_2 : \tau_2,\ \mathtt{C}_2 \\ \tau_1 = z_1{}^{\mathtt{n}_1} \quad \tau_2 = z_2{}^{\mathtt{n}_2} \quad \mathtt{if}.label = \mathtt{b}\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \mathtt{<}\tau_1, \tau_2\mathtt{>}^{\mathtt{n}}, \\ \mathtt{C}_b \cup \mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = (\mathtt{b} \times \mathtt{n}_1 + \mathtt{notb} \times \mathtt{n}_2)\}\end{array}}\,[\textsc{Cond}]$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \\ \Gamma \uplus \{x : \tau_1\} \vdash e_2 : \tau_2,\ \mathtt{C}_2\end{array}}{\Gamma \vdash \mathtt{let}\ x\ =\ e_1\ \mathtt{do}\ e_2 : \tau_2,\ \mathtt{C}_1 \cup \mathtt{C}_2}\,[\textsc{Let}]$$

$$\frac{\begin{array}{c}\Gamma \vdash e_0 : u^1,\ \mathtt{C} \qquad (u'^1 \mid \tau') = split^p(u^1) \\ \Gamma \uplus \{x : u'^1\} \vdash e_1 : \tau_1,\ \mathtt{C}_1 \\ \Gamma \uplus \{x_2 : \tau'\} \vdash e_2 : \tau_2,\ \mathtt{C}_2\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{case}\ e_0\ \mathtt{of}\ x : p \Rightarrow e_1 \mid x_2 \Rightarrow e_2\ \mathtt{end} : \\ ((\tau_1\ \text{if}\ u'^1 \neq \emptyset) \mid (\tau_2\ \text{if}\ \tau' \neq \emptyset))^{\mathtt{n}}, \mathtt{C} \cup \mathtt{C}_1 \cup \mathtt{C}_2\end{array}}\,[\textsc{Case}]$$

Figure 11: Type rules except the rules for the `for` expression.

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \tau_2,\ \mathtt{C}_2 \\ \mathtt{for}.label = \emptyset\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_2, \\ \mathtt{C}_1 \cup \mathtt{C}_2\end{array}}\,[\textsc{For}]$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : (z^{\mathtt{m}})^{*\mathtt{n}},\ \mathtt{C}_2 \\ \mathtt{for}.label = start \qquad \exists\pi.\ \pi \in \mathtt{C}_2\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : (z^{\mathtt{m}})^{*\mathtt{n}}, \\ \mathtt{C}_1 \cup (\mathtt{C}_2 \setminus \{\pi\}) \cup \{\mathtt{n} = \mathtt{m} \times \Gamma_\pi(\pi)\}\end{array}}\,[\textsc{For}\Pi]$$

$$\frac{\Gamma \uplus \{x : u^1\} \vdash e_2 : \tau,\ \mathtt{C} \quad x.label = \emptyset}{\Gamma \uplus \{\mathtt{for}\ x : u^1\} \vdash e_2 : \tau,\ \mathtt{C}}\,[\textsc{ForU}] \qquad \frac{\Gamma \uplus \{x : u^1\} \vdash e_2 : z^{\mathtt{m}},\ \mathtt{C} \quad x.label = \pi}{\Gamma \uplus \{\mathtt{for}\ x : u^1\} \vdash e_2 : (z^{\mathtt{m}}) *^{\mathtt{n}},\ \mathtt{C} \cup \{\pi\}}\,[\textsc{For}\Pi\textsc{U}]$$

$$\frac{}{\Gamma \uplus \{\mathtt{for}\ x : ()^0\} \vdash e_2 : ()^0}\,[\textsc{ForN}]$$

$$\frac{}{\Gamma \uplus \{\mathtt{for}\ x : \emptyset\} \vdash e_2 : \emptyset}\,[\textsc{ForE}]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \tau_1',\ \mathtt{C}_1 \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \tau_2',\ \mathtt{C}_2 \\ \tau_1' = z_1^{\mathtt{m}_1} \quad \tau_2' = z_2^{\mathtt{m}_2} \qquad \neg\exists\pi.\ \pi \in \mathtt{C}_{1,2}\end{array}}{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : (\tau_1, \tau_2)^{\mathtt{n}}\} \vdash e_2 : (\tau_1', \tau_2')^{\mathtt{n}}, \\ \mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = \mathtt{m}_1 + \mathtt{m}_2\}\end{array}}\,[\textsc{ForS}]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \tau_1',\ \mathtt{C}_1 \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \tau_2',\ \mathtt{C}_2 \\ \neg\exists\pi.\ \pi \in \mathtt{C}_{1,2}\end{array}}{\Gamma \uplus \{\mathtt{for}\ x : (\tau_1 | \tau_2)^{\mathtt{n}}\} \vdash e_2 : (\tau_1' | \tau_2')^{\mathtt{n}'},\ \mathtt{C}_1 \cup \mathtt{C}_2}\,[\textsc{ForC}]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \tau_1',\ \mathtt{C}_1 \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \tau_2',\ \mathtt{C}_2 \\ \tau_1' = z_1^{\mathtt{m}_1} \quad \tau_2' = z_2^{\mathtt{m}_2} \qquad \neg\exists\pi.\ \pi \in \mathtt{C}_{1,2} \\ \{\mathtt{n} = \mathtt{b} \times \mathtt{r}_1 + \mathtt{notb} \times \mathtt{r}_2\}\end{array}}{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \mathtt{<}\tau_1, \tau_2\mathtt{>}^{\mathtt{n}}\} \vdash e_2 : \mathtt{<}\tau_1', \tau_2'\mathtt{>}^{\mathtt{n}'}, \\ \mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = \mathtt{b} \times \mathtt{r}_1 + \mathtt{notb} \times \mathtt{r}_2\} \\ \cup\ \{\mathtt{n}' = \mathtt{b} \times \mathtt{m}_1 + \mathtt{notb} \times \mathtt{m}_2\}\end{array}}\,[\textsc{ForP}]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau\} \vdash e_2 : \tau',\ \mathtt{C} \\ \tau = z^{\mathtt{m}} \quad \tau' = z_1^{\mathtt{m}_1} \quad \neg\exists\pi.\ \pi \in \mathtt{C}\end{array}}{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau *^{\mathtt{n}}\} \vdash e_2 : \tau' *^{\mathtt{n}_1}, \\ \{\mathtt{n}_1 = (\mathtt{n}/\mathtt{m}) \times \mathtt{m}_1\} \cup \mathtt{C}\end{array}}\,[\textsc{ForR}]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \tau',\ \mathtt{C} \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \tau',\ \mathtt{C} \\ \exists\pi.\ \pi \in \mathtt{C}\end{array}}{\Gamma \uplus \{\mathtt{for}\ x : \tau\} \vdash e_2 : \tau',\ \mathtt{C}}\,[\textsc{For}\Pi 1]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \tau',\ \mathtt{C} \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : (\,)^0,\ \mathtt{C}' \\ \exists\pi.\ \pi \in \mathtt{C} \quad \tau' \neq (\,)^0\end{array}}{\Gamma \uplus \{\mathtt{for}\ x : \tau\} \vdash e_2 : \tau',\ \mathtt{C}}\,[\textsc{For}\Pi 2]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : (\,)^0,\ \mathtt{C}' \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \tau',\ \mathtt{C} \\ \exists\pi.\ \pi \in \mathtt{C} \quad \tau' \neq (\,)^0\end{array}}{\Gamma \uplus \{\mathtt{for}\ x : \tau\} \vdash e_2 : \tau',\ \mathtt{C}}\,[\textsc{For}\Pi 3]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \tau',\ \mathtt{C} \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \emptyset \\ \exists\pi.\ \pi \in \mathtt{C} \quad \tau' \neq \emptyset\end{array}}{\Gamma \uplus \{\mathtt{for}\ x : \tau\} \vdash e_2 : \tau',\ \mathtt{C}}\,[\textsc{For}\Pi 4]$$

$$\frac{\begin{array}{c}\Gamma \uplus \{\mathtt{for}\ x : \tau_1\} \vdash e_2 : \emptyset \\ \Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \tau',\ \mathtt{C} \\ \exists\pi.\ \pi \in \mathtt{C} \quad \tau' \neq \emptyset\end{array}}{\Gamma \uplus \{\mathtt{for}\ x : \tau\} \vdash e_2 : \tau',\ \mathtt{C}}\,[\textsc{For}\Pi 5]$$

$$\frac{\Gamma \uplus \{\mathtt{for}\ x : \tau\} \vdash e_2 : \tau', \mathtt{C} \quad \exists\pi.\ \pi \in \mathtt{C}}{\Gamma \uplus \{\mathtt{for}\ x : (\tau *)^{\mathtt{n}}\} \vdash e_2 : \tau',\ \mathtt{C}}\,[\textsc{For}\Pi\textsc{R}]$$

Figure 12: Type rules for the `for` expression.

This rule is straightforward: the number of XML elements produced by the sequence expression as a whole is the sum of the numbers of XML elements produced by its subexpressions. The rule adds the constraint that n, the dimension of the sequence expression, is equal to the sum of the dimensions of the subexpressions, namely $m_1 + m_2$.

3.2.2 Conditional Expressions

Our type system allows the `true` and `false` branches of a conditional expression to have different types. The type of the conditional expression in most type systems is a choice type composed of the types of the branches: $(\tau_1|\tau_2)$. The loss of precision from this approach poses a problem: we can determine that the value of the dimension variable for the conditional expression is within the range of the dimensions of its branches, but we can no longer conclude that two dimensions are equal. We address this problem by means of a *pair type*, introduced in Section 3.1, plus some post-processing to relate pair types.

Our type rule for conditional expressions is as follows:

$$\frac{\begin{array}{c}\Gamma \vdash e_b : \texttt{Boolean}^{1},\ \texttt{C}_b \\ \Gamma \vdash e_1 : \tau_1,\ \texttt{C}_1 \qquad \Gamma \vdash e_2 : \tau_2,\ \texttt{C}_2 \\ \tau_1 = z_1^{\texttt{n}_1} \qquad \tau_2 = z_2^{\texttt{n}_2} \qquad \texttt{if}.label = \texttt{b}\end{array}}{\begin{array}{c}\Gamma \vdash \texttt{if } e_b \texttt{ then } e_1 \texttt{ else } e_2 : \texttt{<}\tau_1,\tau_2\texttt{>}^{\texttt{n}}, \\ \texttt{C}_b \cup \texttt{C}_1 \cup \texttt{C}_2 \cup \{\texttt{n} = \texttt{b} \times \texttt{n}_1 + \texttt{notb} \times \texttt{n}_2\}\end{array}}$$

A pre-processing phase labels each conditional expression with a unique label. The hypothesis "$\texttt{if}.label = \texttt{b}$" extracts that label for use in the constraints. The variables b and notb in the conclusion cannot simply be fresh variables because in the post-processing it is necessary to relate the variables to the conditional expression that produces them.

This rule adds a constraint which equates n, the dimension of the pair type, to an expression which includes b and notb. Intuitively these can be thought of as representing the number of times e_b is true and the number of times it is false across all executions of the conditional expression in one run of the program. During type inference, these are considered unbound variables.

The variables b and notb become useful when constraints are added during the post-processing phase that declares them to be equal to another pair of variables, $\texttt{b}'$ and $\texttt{notb}'$, respectively. The post-processing adds these constraints as it puts the boolean expressions (e_b), which these variables name, into equivalence classes. Here are the sufficient conditions for putting two boolean expressions into one equivalence class: First, for any input, there exists a bijective mapping from the `true` evaluations of one boolean expression to the `true` evaluations of the other, and similarly with `false` evaluations. Note that this condition ignores order. Second, if the conditional expressions do occur in the context of iteration, they must both be at the top of that context. For example, a path followed by a conditional fits these criteria.

Our formulation of the post-processing finds syntactically equivalent boolean expressions that have equivalently bound variables and that get executed the same number of times. We first determine which variables are equivalently bound by finding the congruence closure [12] of the variables used in all boolean expressions. However, all variable bindings are not the same. Figure 13 introduces the `for` expression, which changes its variable's binding on each iteration. The static binding must indicate this to avoid confusion with `let` and `case` bindings. We denote the binding of variables from `for` expressions using the keyword `for`. For example, in Figure 13, `x` is bound to `for /book/~`, where `~` is a wildcard that matches any tag, and `x1` is bound to `for /book/title`. Leaving off the `for` in the binding would denote the whole list/tree at once rather than one element at a time.

```
τs = Book = book[ author[ Str ]+,
                  title[ Str ],
                  subtitle[ Str ]?,
                  isbn[ Int ] ]
let book0 : Book
for x in children(book0) do
       case x of
         x1:title => x1
         x2 => ()
       end
τ' =()+, title[ Str ], ()?, ()
  = title[ Str ]
```

Figure 13: The inferred type for the `for` expression.

Next, we establish a criterion for saying a boolean expression can be known to get executed the same number of times as another boolean expression. If a boolean expression does not meet this criterion, we do not consider it further. The criterion is as follows: In order for a variable in a boolean expression (or anywhere else) to have a `for` binding, it must be in the context of a `for` expression. We say that a boolean expression is in its *minimum iteration context* if it is nested within the fewest `for` expressions possible given its variables' bindings. If a boolean expression is in its minimum iteration context, we can guarantee that another boolean expression, whose variables are bound equivalently and which is in its minimum iteration context, gets executed the same number of times. If a boolean expression is in the context of more iterations than required by its variables' bindings, we do not consider it further.

The boolean expressions that remain eligible for equivalence are finally tested for simple structural equivalence, such that variables at corresponding positions must have equivalent bindings. The equivalence classes of structurally equivalent boolean expressions are joined. A more sophisticated analysis could find more equivalences, but since that is not the focus of this work, we do not pursue it further.

3.2.3 Repetition Expressions

The `for` expression is the principle mechanism for producing lists of unspecified length, and is the most involved to type.

In [8], a technique for typing the `for` expression is introduced, in which a type for e_2 is found once for each unique unit type in τ_1 (e_1's type), and those types are composed according to the type structure of τ_1. Before going into detail about our type rules for the `for` expression, we review the main idea of that technique. Consider the source type and program in Figure 13. The type `Book` has a root with tag `book`, which has some children. Those children are one or more `authors`, a `title`, an optional `subtitle`, and an `isbn` number. Each of those tags has a scalar child. The variable `book0` is an instance of `Book`. The program iterates over the children of `book0`, and, in the case when the current child is a `title`, it outputs the child; otherwise it outputs an empty sequence.

The expression `children(book0)` has four unique unit types, so we type the `case` expression (e_2) with each of the unit types successively bound to the iteration variable `x`:

- when `x` has type `author[Str]`, e_2 has type `()`;
- when `x` has type `title[Str]`, e_2 has type `title[Str]`;
- when `x` has type `subtitle[Str]`, e_2 has type `()`;
- when `x` has type `isbn[Int]`, e_2 has type `()`.

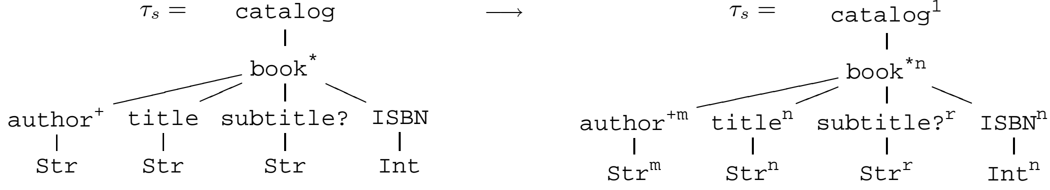

Figure 15: The input type tree annotated.

```
τs = Catalog = catalog[ Book* ]
let cat0 : Catalog
 1 titles[ for w in children(cat0) do
 2   case w of
 3     w1:book =>
 4       for x in children(w1) do
 5         case x of
 6           x1:title => x1
 7           x2 => ()
 8         end
 9     w2 => ()
10   end ],
11 isbns[ for y in children(cat0) do
12   case y of
13     y1:book =>
14       for z in children(y1) do
15         case z of
16           z1:isbn => z1
17           z2 => ()
18         end
19     y2 => ()
20   end ]
```

Figure 14: A program that makes a list of all titles followed by all ISBNs from a catalog.

When we replace each of these unit types in τ_1 with the corresponding inferred types for e_2, we get τ', as shown in Figure 13. After doing some simplification to get rid of the empty sequence types, we get τ' = `title[Str]`. This is accomplished by means of auxiliary rules, similar to the auxiliary rules in our type system.

Figure 14 shows a program fragment that we will use as an example for explaining our treatment of the `for` expression. The type `Book` is as defined in Figure 13. The sequences of nested `for` and `case` expressions have the same semantics as the paths in Figure 1, so with the exception of the missing outermost `doc` element, Figure 14 has the same semantics as Figure 1. Because each `book` has exactly one `title` and one `isbn`, we expect `titles` and `isbns` to have the same number of children.

Source Type Annotation If two expressions must produce lists of the same length, unless the output is constant and static, it is because the expressions are related in the same way to the input type. More specifically, they both iterate over input-tree-paths of equal size, and they both output the same number of unit values per iteration. In order to follow this intuition, we annotate the source type in a pre-processing phase. The source type pre-processing is a top-down, breadth-first process. Our goal is to annotate each path with the number of unit values found by following that path in the input tree. If two paths specify the same number of unit values, their annotations should be the same. Figure 15 shows the input type as a tree before and after the annotation. Consider the type trees as having levels, and the trees in Figure 15 are drawn such that the levels are distinguished by height with the first level on top. When level i has been annotated, we annotate level $i + 1$ as follows: For the elements `a` at the end of path π/`a` annotated with `n`, if all have exactly j children that match a pattern p, then all such children are annotated with j×`n` (or simply `n` if j is 1). If this relationship does not hold, then the children matching pattern p are given a fresh annotation. In Figure 15, `book` has exactly one child `title`, so `title` receives the same annotation as book. The number of `author`'s each `book` has is not specified, so `author` receives a fresh annotation.

These annotations are not dimension annotations. However, the annotations on the inferred type are dimension annotations. This creates an input/output incongruity. Indeed, we would like to avoid such incongruities since an intermediate result within an XQuery program may become the input to another part of the same program. However, we have argued in Section 1.1 that basing inferences on the number of unit values in a source path enables a natural type-based analysis. Additionally, while input type tree annotation according to paths is top-down and breadth-first, type inference is bottom-up and modular, so traditional type systems are not well-suited to the task of annotating inferred results according to paths. Consequently, this incongruity, though not desirable, is by design. In the case of output based on intermediate results, we do not risk unsoundness because expressions that iterate over intermediate results will not be annotated (by the program annotation in the next section), so some information will not propagate through.

Program Annotation Next, a pre-processing phase on the program is executed. The purpose of this pre-processing is to find and annotate iteration expressions that can be related directly back to τ_s. Our formulation of the pre-processing finds nested `for` and `case` expressions which have the same structure as could be produced by the translation of a path into XQuery Core. Figure 16 gives the algorithm for this pre-processing. It is called by passing to it the root of the abstract syntax tree of the given program. In Figure 14, this structure appears twice: first in lines `1` through `10` and then again in lines `11` through `20`. The first half corresponds to the path `/books/book/title` and the second half corresponds to `/books/book/isbn`. The pre-processing automatically adds annotations to the program that will be used by the type system to infer sizes that can be related to each other. For each path found, the `for` at the start is labeled $start$, and the iteration variable at the end is labeled with the path π. For the program in Figure 14, the `for`'s on lines `1` and `11` are labeled $start$, `x` on line `4` is labeled `/books/book/title`, and `z` on line `14` is labeled `/books/book/isbn`.

Typing Repetition Expressions Figure 12 gives the full list of rules and auxiliary rules for typing the `for` expression. The rules [FOR] and [FORΠ] type `for` expressions, and the rest are auxiliary rules. When one of the hypotheses includes a type assignment

```
FINDPATH(node n)
 1  if n.type = for and n.child[1] = root
 2    then c2 ← n.child[2]
 3         n.label ← start
 4         if c2.type = for
 5            and c2.child[1].type = children
 6            and c2.child[1].child[0] = n.child[0]
 7           then FOLLOWPATH(c2, ε)
 8           else n.child[0].label ← /
 9    else for each n′ in children(n)
10         do FINDPATH(n′)

FOLLOWPATH(node n, path p)
 1  c2 ← n.child[2]
 2  if c2.type = case
 3    then d3 ← c2.child[3]
 4    else n.child[0].label ← p/~
 5         return
 6  if n.child[0].name = c2.child[0].name
 7    then p ← p/c2.child[2].name
 8    else n.child[0].label ← p/~
 9         return
10  if d3.type = for and c2.child[5].type = ()
11    then e1 ← d3.child[1]
12    else n.child[0].label ← p
13         return
14  if e1.type = children
15     and e1.child[0].name = c2.child[1].name
16    then FOLLOWPATH(d3, p)
17    else n.child[0].label ← p
18         return
```

Figure 16: Pre-processing on program: label the sequence of `for` expressions that follow a path through the input tree with the path followed. The `for` expression has the form (`for` x `in` e_1 `do` e_2), so syntax tree nodes of type `for` have 3 children: x, e_1, and e_2. The `case` expression has the form (`case` e `of` x_1`:p =>` e_1 `|` x_2 `=>` e_2), so `case` nodes have 6 children: e, x_1, `p`, e_1, x_2, and e_2.

of the form $\{\texttt{for}\ \ x : \tau\}$, the auxiliary rules resolve the type of the expression in that hypothesis. The hypothesis "$\texttt{for}.label = \ldots$" in the hypotheses of [FOR] and [FORΠ] refers to the result of program annotation from last paragraph: if the `for` expression is the first expression in a sequence that represents a path, it is labeled *start*; otherwise it has no label. The same is true of the "$x.label = \ldots$" in [FORU] and [FORΠU]: if the `for` expression is the last `for` expression in a sequence that represents a path, the iteration variable is labeled with the path it represents; otherwise it has no label. In the rules [FORΠ1] through [FORΠ5], τ can be instantiated to (τ_1, τ_2), $(\tau_1 | \tau_2)$, or $<\tau_1, \tau_2>$. The hypothesis $\neg\exists\pi.\ \pi \in \mathtt{C}_{1,2}$ in [FORS] through [FORP] means that there exists no path constraints in the constraint sets of the preceding hypotheses. Conversely, in [FORΠ1] through [FORΠ5], $\exists\pi.\ \pi \in \mathtt{C}$ means there is a path constraint in the constraint set `C`.

We will now explain these rules using the example program in Figure 14. Figure 17 shows the type derivation for the `for` expression on lines `1–10` of Figure 14. The expressions, environments, and types named in the derivation are shown directly below the derivation tree. The type τ_s is the source type, the path π is the path this expression iterates over, τ' is the inferred type, and e_a is the `for` expression on lines `1–10`.

The `x` on line `5` of Figure 14 iterates over the values represented by the type of the children of `book` (the type tree drawn in Figure 15 shows this type as `author[...]`$^+$, `title[...]`^{1}...). The derivation labeled '①' is the root of the uses of auxiliary rules for the `for` expression to decompose this type according to its structure. The derivation branches on the left decompose the first half (`author`$^+$, `title`^{1}, or equivalently `author`^{1}, `author`*, `title`^{1}) and the branches on the right decompose the rest (abbreviated with '≀'). We illustrate the use of the rule [FORΠU] with the derivation labeled '②'. This derivation determines the type for the body of the inner `for` expression when `x` has type `title[String`^{1}`]`^{1}. As determined by the [CASE] rule, the `case` expression has type `title[String`^{1}`]`^{1}. It may seem surprising that ② says that when `x` has the unit type `title` and the `case` expression has the same type, the body of the `for` expression is $\tau' = \texttt{title[String}^1\texttt{]}^{1*\mathtt{r}}$. Although the data produced by the `case` expression will always fall into type τ', τ' is less precise than it could be. However, notice that both τ' and π are propagated down from ② to the root of the derivation tree. As the root of the tree, they are linked together with the constraint $\mathtt{r} = 1 \times \Gamma_\pi(\pi)$. This constraint is valid because the number of unit values produced across all iterations of the outer `for` expression over a path equals the number produced in one iteration (1) times the number of iterations. In the intermediate stages of the derivation, it may not be possible to say precisely how many elements are produced by the particular sub-expression being typed, so we say "there is an unknown number" by leaving `r` unbound until the derivation of the outermost `for` expression.

The derivation immediately below ②, using the rule [FORΠ3], decomposes a sequence type (τ_{s2a}). It may seem that the two types of e_{2b} from the hypotheses should be sequenced as the type of e_{2b} in the conclusion. However, the goal is to carry the type derived from ② to the root, so that type goes into the conclusion, and the $()^0$ gets ignored. Similarly, the derivation using the rule [FORΠR] right above the root derivation decomposes a starred type. The type of e_{2a} in the conclusion is not a starred version of the type of e_{2a} in the hypothesis; it is the same type. Even though a starred version of the type would be semantically equivalent, the rule [FORΠ] grabs the dimension variable that is nested within the outermost star as the number of unit values produced in a single iteration. If another star were added, the dimension variable nested below the outermost star would be an unbound variable, and the constraint produced by [FORΠ] would not constrain the dimension of τ' at all.

The remainder of the auxiliary rules ([FORN] through [FORR]) follow the pattern of the auxiliary rules given in [8] with the exception of [FORP], since the type system in [8] does not include a pair type. The structure of a type derivation tree using these rules would be identical to the structure of one using the rules [FORΠn] (see, for example, Figure 17). Unlike the rules [FORΠn], these auxiliary rules do reconstruct the structure of the type they decompose. For example, the type inferred using [FORS] would be the sequence of the types in [FORS]'s hypotheses.

3.2.4 Case Expressions

The type rule [CASE] relies on a *split* function. In Section 2.2 the semantics of matching a value against a pattern was given. The *split* function follows that semantics in a type setting: $split^p(\tau) = (u^1 | \tau')$ such that if all values of type τ match pattern p, $u^1 = \tau$ and $\tau' = \emptyset$; and if no values of type τ match pattern p, $u^1 = \emptyset$ and $\tau' = \tau$. The complete rules for *split* are given in Figure 18.

The [CASE] rule produces a choice type with an unbound dimension variable. However, the unbound dimension variable does not prevent dimension inference because in all cases but one, either

$$\dfrac{\dfrac{\Gamma_6(x)=\tau_{s2g}}{\Gamma_6 \vdash x:\tau_{s2g}}\text{VAR} \quad (\emptyset|author)=split^{title}(author) \quad \dfrac{\Gamma_7(x1)=\emptyset}{\Gamma_7 \vdash x1:\emptyset}\text{VAR} \quad \dfrac{}{\Gamma_8 \vdash ():()^0}\text{NIL}}{\Gamma_6 \vdash e_2 : ()^0}\text{CASE}$$

$$\vdots$$

$$\dfrac{\dfrac{x.label=\pi}{\Gamma_2 \uplus \{\texttt{for}\, x:\tau_{s2g}\} \vdash e_{2b} : ()^0, \{\pi\}}\text{FOR}\Pi\text{U}}{\Gamma_2 \uplus \{\texttt{for}\, x:\tau_{s2h}\} \vdash e_{2b} : ()^0, \{\pi\}}\text{FOR}\Pi\text{R}$$

$$\dfrac{\Gamma_6(x)=\tau_{s2g}}{\Gamma_6 \vdash x:\tau_{s2g}}\text{VAR} \quad (\emptyset|author)=split^{title}(author) \quad \dfrac{\Gamma_7(x1)=\emptyset}{\Gamma_7 \vdash x1:\emptyset}\text{VAR} \quad \dfrac{}{\Gamma_8 \vdash ():()^0}\text{NIL}$$
$$\overline{\Gamma_6 \vdash e_2 : ()^0}\ \text{CASE}$$

$$\vdots$$

$$\dfrac{\dfrac{x.label=\pi}{\Gamma_2 \uplus \{\texttt{for}\, x:\tau_{s2g}\} \vdash e_{2b} : ()^0, \{\pi\}}\text{FOR}\Pi\text{U} \qquad \vdots}{\Gamma_2 \uplus \{\texttt{for}\, x:\tau_{s2c}\} \vdash e_{2b} : ()^0, \{\pi\}}\text{FOR}\Pi 1$$

$$\dfrac{\Gamma_3(x)=\tau_{2sd}}{\Gamma_3 \vdash x:\tau_{2sd}}\text{VAR} \quad (\tau_{s2d}|\emptyset)=split^{title}(\tau_{s2d}) \quad \dfrac{\Gamma_4(x1)=\tau_{s2d}}{\Gamma_4 \vdash x1:\tau_{s2d}}\text{VAR} \quad \dfrac{}{\Gamma_5 \vdash ():()^0}\text{NIL}$$
$$\overline{\Gamma_3 \vdash e_{2b} : \texttt{title[String}^1\texttt{]}^1}\ \text{CASE}$$

$$\vdots$$

$$\dfrac{\vdots \qquad \dfrac{x.label=\pi}{\Gamma_2 \uplus \{\texttt{for}\, x:\tau_{2sd}\} \vdash e_{2b} : \tau', \{\pi\}}\text{FOR}\Pi\text{U} ❷}{\Gamma_2 \uplus \{\texttt{for}\, x:\tau_{2sa}\} \vdash e_{2b} : \tau', \{\pi\}}\text{FOR}\Pi 3$$

$$\wr\ \text{- FOR}\Pi\text{U} \quad \wr\ \text{- FOR}\Pi\text{U} \quad \wr\ \text{FOR}\Pi 1 \quad \wr\ \text{- FOR}\Pi\text{U} \quad \wr\ \text{FOR}\Pi 1$$

$$\dfrac{\texttt{for}.label=\emptyset \quad \dfrac{\dfrac{\Gamma_2(w1)=\tau_{s1a}}{\Gamma_2 \vdash w1:\tau_{s1a}}\text{VAR}}{\Gamma_2 \vdash e_{1b}:\tau_{s2}}\text{CHILD} \quad \dfrac{\vdots \qquad \wr}{\Gamma_2 \uplus \{\texttt{for}\, x:\tau_{s2}\} \vdash e_{2b} : \tau', \{\pi\}}\text{FOR}\Pi 2 ❶}{\Gamma_1 \vdash e_b : \tau', \{\pi\}}\text{FOR}$$

$$\dfrac{\dfrac{\Gamma_1(w)=\tau_{s1a}}{\Gamma_1 \vdash w:\tau_{s1a}}\text{VAR} \quad (\tau_{s1a}|\emptyset)=split^{book}(\tau_{s1a}) \quad \vdots \quad \dfrac{}{\Gamma_1 \uplus \{w2:\emptyset\} \vdash ():()^0}\text{NIL}}{\Gamma_1 \vdash e_{2a} : \tau', \{\pi\}}\text{CASE}$$

$$\vdots$$

$$\dfrac{\texttt{for}.label=start \quad \dfrac{\dfrac{\Gamma_0(cat0)=\tau_s}{\Gamma_0 \vdash cat0:\tau_s}\text{VAR}}{\Gamma_0 \vdash e_{1a}:\tau_{s1}}\text{CHILD} \quad \dfrac{\dfrac{w.label=\emptyset}{\Gamma_0 \uplus \{\texttt{for}\, w:\tau_{s1a}\} \vdash e_{2a} : \tau', \{\pi\}}\text{FOR1}}{\Gamma_0 \uplus \{\texttt{for}\, w:\tau_{s1}\} \vdash e_{2a} : \tau', \{\pi\}}\text{FOR}\Pi\text{R}}{\Gamma_0 \vdash e_a : \tau', \{\texttt{r = 1} \times \Gamma_\pi(\pi)\}}\text{FOR}\Pi$$

$e_a =$ `for w in` e_{1a} `do` e_{2a}
$e_{1a} =$ `children(cat0)`
$e_{2a} =$ `case w of w1:book =>` e_b `| w2 => () end`
$e_b =$ `for x` e_{1b} `do` e_{2b}
$e_{1b} =$ `children(w1)`
$e_{2b} =$ `case x of x1:title => x1 | x2 => () end`
$\Gamma_0 = \{\texttt{cat0:}\tau_s\}$
$\Gamma_1 = \Gamma_0 \uplus \{\texttt{w:}\tau_{s1a}\}$
$\Gamma_2 = \Gamma_1 \uplus \{\texttt{w1:}\tau_{s1a}\}$
$\Gamma_3 = \Gamma_2 \uplus \{\texttt{x:}\tau_{s2d}\}$ $\quad \Gamma_6 = \Gamma_2 \uplus \{\texttt{x:}\tau_{s2g}\}$
$\Gamma_4 = \Gamma_3 \uplus \{\texttt{x1:}\tau_{s2d}\}$ $\quad \Gamma_7 = \Gamma_6 \uplus \{\texttt{x1:}\emptyset\}$
$\Gamma_5 = \Gamma_3 \uplus \{\texttt{x2:}\emptyset\}$ $\quad \Gamma_8 = \Gamma_6 \uplus \{\texttt{x2:}\tau_{s2g}\}$

$\tau_s = \texttt{catalog[}\ \tau_{s1}\ \texttt{]}^1$
$\tau_{s1} = \tau_{s1a}{}^{*q}$
$\tau_{s1a} = \texttt{book[}\ \tau_{s2}\ \texttt{]}^1$
$\tau_{s2} = (\tau_{s2a}, \tau_{s2b})^{p}$
$\tau_{s2a} = (\tau_{s2c}, \tau_{s2d})^{\ell}$
$\tau_{s2c} = (\tau_{s2g}, \tau_{s2g}{}^{*k})^{j}$
$\tau_{s2g} = \texttt{author[String}^1\texttt{]}^1$
$\tau_{s2d} = \texttt{title[String}^1\texttt{]}^1$
$\tau_{s2b} = (\tau_{s2e}, \tau_{s2f})^{n}$
$\tau_{s2e} = \texttt{<subtitle[String}^1\texttt{]}^1\texttt{,()}^0\texttt{>}^{m}$
$\tau_{s2f} = \texttt{isbn[Integer}^1\texttt{]}^1$

$\tau' = \texttt{title[String}^1\texttt{]}^{1*r}$
$\pi = \texttt{/catalog/book/title}$

Figure 17: Type derivation for lines `1–10` of Figure 14.

$$
\begin{array}{lll}
split^{a}(s) & = a[\emptyset] \mid s & \\
split^{a}(a[\tau]) & = a[\tau] \mid \emptyset & \\
split^{a}(a'[\tau]) & = a[\emptyset] \mid a'[\tau] & \text{if } a \neq a' \\
split^{a}(\sim[\tau]) & = a[\tau] \mid \sim[\tau] & \\
split^{a}(\tau_1 \mid \tau_2) & = a[\tau_1' \mid \tau_2'] & \text{where} \\
 & \quad \mid (\tau_1'' \mid \tau_2'') & a[\tau_i'] \mid \tau_i'' = split^{a}(\tau_i) \\
split^{a}{<}\tau_1, \tau_2{>} & = a[\tau_1' \mid \tau_2'] & \text{where} \\
 & \quad \mid {<}\tau_1'', \tau_2''{>} & a[\tau_i'] \mid \tau_i'' = split^{a}(\tau_i) \\
split^{a}(\emptyset) & = a[\emptyset] \mid \emptyset & \\
 & & \\
split^{\sim}(s) & = \sim[\emptyset] \mid s & \\
split^{\sim}(a[\tau]) & = \sim[\tau] \mid \emptyset & \\
split^{\sim}(\sim[\tau]) & = \sim[\tau] \mid \emptyset & \\
split^{\sim}(\tau_1 \mid \tau_2) & = \sim[\tau_1' \mid \tau_2'] & \text{where} \\
 & \quad \mid (\tau_1'' \mid \tau_2'') & \sim[\tau_i'] \mid \tau_i'' = split^{\sim}(\tau_i) \\
split^{\sim}{<}\tau_1, \tau_2{>} & = \sim[\tau_1' \mid \tau_2'] & \text{where} \\
 & \quad \mid {<}\tau_1'', \tau_2''{>} & \sim[\tau_i'] \mid \tau_i'' = split^{\sim}(\tau_i) \\
split^{\sim}(\emptyset) & = \sim[\emptyset] \mid \emptyset & \\
 & & \\
split^{s}(s') & = s' \mid \emptyset & \text{if } s' = s \\
split^{s}(a[\tau]) & = \emptyset \mid a[\tau] & \\
split^{s}(\sim[\tau]) & = \emptyset \mid \sim[\tau] & \\
split^{s}(\tau_1 \mid \tau_2) & = (s_1 \mid s_2) & \text{where} \\
 & \quad \mid (\tau_1' \mid \tau_2') & s_i \mid \tau_i' = split^{s}(\tau_i) \\
split^{s}{<}\tau_1, \tau_2{>} & = {<}s_1, s_2{>} & \text{where} \\
 & \quad \mid {<}\tau_1', \tau_2'{>} & s_i \mid \tau_i' = split^{s}(\tau_i) \\
split^{s}(\emptyset) & = \emptyset \mid \emptyset &
\end{array}
$$

Figure 18: Rules for the $split$ function.

$u^1 = \emptyset$ or $\tau' = \emptyset$. That one exception occurs when the variable has type $\sim$ and the pattern is a tag: some values will match the pattern and some will not. In this case our analysis is unable to infer dimensional relationships for this expression. However, such a construct leads to an invalid path, for which our input type annotations will not help. By refining the definition of a "valid path" and the input type annotations, we can have a more precise analysis. The inferred type in the hypothesis that has a variable of type $\emptyset$ in its environment drops out from the inferred type in the conclusion. The remaining type has the same constraints as it has in the hypothesis.

3.3 Type Soundness

In this section, we state a soundness theorem for our type system. The proof of the theorem is through subject reduction, following the style of Wright and Felleisen [24].

THEOREM 1 (SUBJECT REDUCTION). *Given the predicates provided by the pre- and post-processing, we have: If* $\Gamma \vdash e : \tau, \mathtt{C}$ *and* $E \vdash e \Downarrow v$ *then* $\Gamma \vdash v : \tau, \mathtt{C}$.

The full proof of this theorem is given in Appendix C. This theorem relies on a few lemmas. The first is a substitution lemma, which is needed to prove soundness for the expressions that bind variables, namely `for`, `case`, and `let`:

LEMMA 1.1 (SUBSTITUTION). *If* $\Gamma \vdash v : \tau, \mathtt{C}$ *and* $\Gamma \uplus \{x : \tau\} \vdash e : \tau', \mathtt{C}'$ *then* $\Gamma \vdash e\,[v/x] : \tau', \mathtt{C} \cup \mathtt{C}'$.

The second lemma is the case lemma. Because the `case` expression relies on the $split$ function, it is necessary to prove that the $split$ function conservatively matches the semantics of the case expression. This proof is also by structural induction, but because the `case` expression only operates on unit values, the inductive hypothesis is only used when τ is a pair or choice type.

LEMMA 1.2 (CASE). *Assume* $\Gamma \vdash v : \tau, \mathtt{C}$ *and* $split^{p}(\tau) = u'^{1} | \tau'$. *If* $v \in \mathrm{Dom}(p)$ *then* $\Gamma \vdash v : u'^{1}, \mathtt{C}$, *and if* $v \notin \mathrm{Dom}(p)$ *then* $\Gamma \vdash v : \tau', \mathtt{C}$.

4. RELATED WORK

In this section, we survey some closely related work.

4.1 Automata-based Techniques

In [18], six ways of representing XML types (including schemas) are classified based on their expressiveness. The types we work with here are regular expression tree types, which are at least as expressive as the six in [18]. In addition, we add dimensional information.

The asymptotic complexity of type checking XML transformation programs, modeled as top-down uniform tree transducers, is studied in [15, 19]. By varying certain parameters, the type checking problems range from polynomial-time complexity to exponential-time complexity.

Perhaps the most significant automata-based work on XML type checking is [17]. It uses a generalization of traditional top-down regular tree transducers called k-pebble tree transducers to demonstrate the decidability of type checking for the broad range of queries that can be expressed by these automata. This technique was applied to a subset of XSLT in [22] for backward type inference. However, adding dimensions to types can lead to non-regular or even non-context-free languages and make type checking undecidable, so automata-based techniques may not be well-suited for our purposes.

4.2 Type System-based Techniques

In contrast to the previous automata-based approaches, XQuery relies on a language-based technique, namely that of a type system, to accomplish XML type checking [3]. XQuery's type system assigns a type to each expression, such that the XML data created by that expression conforms to the expression's type.

The most interesting part of the XQuery language in terms of types is the `for` expression. The `for` expression has the form (`for` x `in` e_1 `do` e_2), and its operational semantics are essentially the same as the semantics in Section 2.2. XQuery's type system types the `for` expression by finding τ_1 as the type for e_1, and then assigning the union of the unit types in τ_1 to x in the typing of e_2. This method of typing the `for` expression has the advantage that e_2 must be typed only once, but it has the disadvantage that the inferred type is somewhat imprecise. For example, XQuery's type system would infer $\tau' =$ `title[Str]*` for the program in Figure 13. Another type system proposed for XQuery introduced the use of auxiliary rules to type `for` expressions more precisely [8]. We adopt this approach in our type system. They would infer $\tau' =$ `title[Str]` for the program in Figure 13. However, they do not attempt to infer any dimension information beyond what the regular type constructors (*e.g.*, *, ?) clearly imply.

Other work on XML type checking is aimed at integrating XML into general-purpose programming languages. In [14], the general-purpose language used is Java, and the work relies on JWIG [5], an extension of Java. XOBE [13] is also an extension of Java with a similar goal, but it is different from [14] in that XML trees in XOBE can only be constructed bottom-up, as opposed to allowing named gaps that can be filled in any order. Castor [10] and JAXB [16] use Java to generate an object model of XML documents from XML

schema. This allows for a higher level of abstraction than working with the document's tree structure directly.

4.3 Size Analysis

We view size analysis as seeking to make claims about the sizes of data structure and other closely-related aspects of programs. The study of size analysis can be seen as starting with the inference of linear constraints for imperative languages [6]. This abstract interpretation-based approach inferred linear relationships among variables without the aid of programmer-provided inductive assertions or human interaction. This topic has significance to logic programming in the sense that inferring bounds on argument sizes can ensure termination [20].

Type-based size analyses relate more closely to our work. One such analysis uses dependent types [25, 26]. They use parameterized types to infer lengths of lists. The parameters can be constrained with linear equalities and inequalities to determine size relationships. Unlike our type system, theirs requires user annotations. Hughes, Pareto, and Sabry type check recursive data structures with size information in the context of a lazy functional language [11]. Chin and Khoo build on this approach by inferring sizes for recursive functions in the context of strict functional languages [4]. They define the size of a function as both a relation between input and output parameters, and invariants of input parameters across recursive calls. They infer sizes in terms of array lengths, tree heights, and integer values. All of these previous approaches only infer flat sizes; even when sizes for trees are inferred, it is in terms of their one-dimensional height. We infer sizes for the richer tree structure and take into account the levels of the subtrees.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a type system for XQuery to infer dimensional relationships within the output type. Our approach relies on annotating paths in the input type as opposed to dimensions on types directly. Because the dimensional relationships are based on a connection between the program and the source type, the programmer could annotate the source type with dimensional relationships known to hold across all input documents. Such relationships would then propagate through to the inferred output type.

There are a few possible directions for future work. In this work we have not considered functions or recursive types. However, we expect that our technique could be extended to include these language constructs. In the case of functions, for better precision, we would need a context-sensitive analysis. Precise handling of recursive functions seems challenging, but ideas based on context-free language reachability may be applied. The challenge with recursive types lies in proper source type annotation. We believe this could be accomplished by annotating the states of the finite automaton that represents a recursive type. Finally, it would be interesting to implement our analysis to gain some practical experiences.

Acknowledgments

We would like to thank Mary Fernandez for helpful answers to our questions on XQuery.

6. REFERENCES

[1] A. Aiken, E. L. Wimmers, and T. K. Lakshman. Soft typing with conditional types. In *POPL*, pages 163–173, 1994.

[2] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). February 1988. URL: `http://www.w3.org/TR/1998/REC-xml19980210`.

[3] D. Chamberlin and *et al.* XQuery 1.0: An XML Query Language. June 2001. URL: `http://www.w3.org/TR/xquery`.

[4] W. Chin and S. Khoo. Calculating sized types. In *PEPM*, pages 62–72, 1999.

[5] A. S. Christensen and A. Møller. JWIG user manual, June 2002. URL: `http://www.brics.dk/JWIG/manual/`.

[6] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, pages 84–96, 1978.

[7] J. Clark (eds.). XML transformations (XSLT) version 1.0. *W3C*, Nov. 1999. URL: `http://www.w3.org/TR/xslt`.

[8] M. Fernandez, J. Siméon, and P. Wadler. An algebra for XML Query. In *FST TCS*, pages 11–45, December 2000.

[9] M. Fitzgerald. Relaxer tutorial, 2003. URL: `http://www.relaxer.org/doc/tutorial/tutorial.html`.

[10] Exolab Group. Castor, 2002. URL: `http://castor.exolab.org`.

[11] J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *POPL*, 1996.

[12] P. C. Kanellakis and P. Z. Revesz. On the relationship of congruence closure and unification. *Journal of Symbolic Computation*, 7(3/4):427–444, 1989.

[13] M. Kempa and V. Linnemann. Type checking in XOBE. In *BTW '03*, pages 227–246, February 2003.

[14] C. Kirkegaard, A. Møller, and M. I. Schwartzbach. Static analysis of xml transformations in java. URL: `http://citeseer.nj.nec.com/593778.html`.

[15] W. Martens and F. Neven. Typechecking top-down uniform unranked tree transducers. In *ICDT 2003*, pages 64–78, 2003.

[16] Sun Microsystems. JAXB, 2002. URL: `http://java.sun.com/xml/jaxb`.

[17] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *PODS*, pages 11–22, 2000.

[18] M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.

[19] T. Pankowski. Transformation of XML data using an unranked tree transducer. In *EC-Web*, pages 259–269, 2003.

[20] D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, pages 199–260, 1994.

[21] D. Suciu. The XML typechecking problem. *SIGMOD Record*, March 2002.

[22] A. Tozawa. Towards static type checking for xslt. In *Document Eng*, pages 18–27, 2001.

[23] M. Wallace and C. Runciman. Haskell and XML: Generic combinators or type-based translation? In *ICFP*, pages 148–159, 1999.

[24] A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, 1994.

[25] H. Xi and F. Pfenning. Eliminating array bound checking through dependent types. In *PLDI*, pages 249–257, 1998.

[26] H. Xi and F. Pfenning. Dependent types in practical programming. In *POPL*, pages 214–227, 1999.

APPENDIX

A. SOME RULES TO MATCH DATA TO TYPES

$$\mathcal{F}_{\text{REP}} :: \frac{\vdash d : \tau \quad \vdash d' : \tau{*}^{\mathtt{m}}}{\vdash d, d' : \tau{*}^{\mathtt{n}}}$$

$$\mathcal{F}_{\text{REP()}} :: \frac{}{\vdash \mathtt{()} : \tau{*}^{\mathtt{n}}}$$

$$\mathcal{F}_{\text{SEQ}} :: \frac{\vdash d : z^{\mathtt{m1}} \quad \vdash d' : z^{\mathtt{m2}}}{\vdash d\,\mathtt{,}\,d' : (z^{\mathtt{m1}}, z^{\mathtt{m2}})^{\mathtt{m1+m2}}}$$

B. SOME SUBSUMPTION RULES

$$\mathcal{S}_{\text{CHC1}} :: \tau_1 <: (\tau_1|\tau_2)^{\mathtt{n}}$$

$$\mathcal{S}_{\text{CHC2}} :: \tau_2 <: (\tau_1|\tau_2)^{\mathtt{n}}$$

$$\mathcal{S}_{\text{PR1}} :: \tau_1 <: \mathtt{<}\tau_1, \tau_2\mathtt{>}^{\mathtt{n}}$$

$$\mathcal{S}_{\text{PR2}} :: \tau_2 <: \mathtt{<}\tau_1, \tau_2\mathtt{>}^{\mathtt{n}}$$

$$\mathcal{S}_{\text{CHCQ}} :: z^{\mathtt{m}} = (z^{\mathtt{m}}|z^{\mathtt{m}})^{\mathtt{m}}$$

$$\mathcal{S}_{\text{EQ}} :: \tau_1 <: \tau_2 \ \wedge\ \tau_2 <: \tau_1 \ \Rightarrow\ \tau_1 = \tau_2$$

$$\mathcal{S}_{\text{NIL}} :: (()^{0}, z^{\mathtt{n}})^{\mathtt{n}} = (z^{\mathtt{n}}, ()^{0})^{\mathtt{n}} = z^{\mathtt{n}}$$

$$\mathcal{S}_{\text{NREP}} :: ()^{0} = (()^{0})^{*\mathtt{n}}$$

$$\mathcal{S}_{\text{UREP}} :: \tau <: (\tau){*}^{\mathtt{n}}$$

$$\mathcal{S}_{\text{CREP}} :: (\tau|\tau)^{\mathtt{m}} <: (\tau){*}^{\mathtt{n}}$$

$$\mathcal{S}_{\text{SREP}} :: (\tau, \tau)^{\mathtt{m}} <: (\tau){*}^{\mathtt{n}}$$

$$\mathcal{S}_{\text{RREP}} :: (\tau)\,{*}^{\mathtt{m}}\ \mathtt{,}\ \ \{\mathtt{m}\ \mathtt{=}\ f\} <: (\tau){*}^{\mathtt{n}}$$

$$\mathcal{S}_{\text{RNST}} :: (\tau)^{*\mathtt{m}} = ((\tau)^{*\mathtt{m}})^{*\mathtt{n}}$$

$$\mathcal{S}_{\text{NREP}} :: z^{\mathtt{m}} = (z^{\mathtt{m}})^{*\mathtt{n}}\mathtt{,}\ \ \{\mathtt{n}\ =\ \mathtt{m}\}$$

$$\mathcal{S}_{\text{EMT}} :: \tau, \emptyset = \emptyset, \tau = \tau$$

C. PROOF OF SOUNDNESS

We prove soundness by induction on $e \Downarrow v$. We want to prove that $(\Gamma \vdash e : \tau \ \wedge \ E \vdash e \Downarrow e') \ \Rightarrow \ \Gamma \vdash e' : \tau$. We will call this proposition IH_{GEN}, and it will be the induction hypothesis for the judgments "above the line."

Paths take the form $\pi = \texttt{/root/a}_1\texttt{/.../a}_n$. Preprocessing predicates:

pred1:

$\text{H}[\text{H}'[\texttt{for } x_n \texttt{ in } e_1 \texttt{ do } e_2]] \wedge x_n.label = \pi \Leftrightarrow$

$\text{H}[\texttt{for } x_1 \texttt{ in } {e_1}' \texttt{ do } \text{H}''[e_2]\,] \wedge \texttt{for}.label = start \Leftrightarrow$

```
H[ for x_1 in children(root) do
      case x_1 of:
        x_1':a_1 =>
          for x_2 in children(x_1') do
              ...
              case x_{n-1} of
                x_{n-1}':a_{n-1} =>
                  for x_n in children(x_{n-1}') do e_2
                x_{n-1}'' => ()
              end
              ...
        x_1'' => ()
      end ]
```

$\wedge\, x_i.label = \emptyset$ for $i \in [1..n-1]$

$\wedge\, \texttt{for } x_i \texttt{...} \Leftrightarrow \texttt{for}.label = \emptyset$ for $i \in [2..n]$

pred2: the `if`'s in all conditional expressions are uniquely labeled, and no label begins with "`not`"

def1: for π as defined in pred1, let $\Gamma_\pi(\pi) = |s|$, where p is as defined in pred2 and $s = \{y \mid d \mapsto x \wedge x \in p\} = \{d \mid d$ in input $\wedge\, d.name = \texttt{a}_n \wedge \ldots \wedge = (d.parent^n).name = \texttt{root}\}$

In this proof, $\mathcal{D}$ is a derivation and $\mathcal{E}$ is an evaluation.

Proof 1. $\mathcal{D}_{[\text{For}]}$		
1 :	$(\Gamma \uplus \{\texttt{for}\, x : \tau_1\} \vdash e_2 : \tau_2\texttt{, } \texttt{C}_2)$ $\Rightarrow ((\Gamma \vdash e_1 : \tau_1\texttt{, } \texttt{C}_1 \wedge E : e_1 \Downarrow v_1$ $\Rightarrow \Gamma \vdash v_1 : \tau_1\texttt{, } \texttt{C}_1)$ $\Rightarrow ((\Gamma \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 : \tau_2\texttt{,}$ $\texttt{C}_1 \cup \texttt{C}_2$ $\wedge\, E : \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2\texttt{, } \texttt{C}_1 \cup \texttt{C}_2))$	by the $\texttt{for}_{\text{AUX}}$ lemma
2 :	$(\Gamma \uplus \{\texttt{for}\, x : \tau_1\} \vdash e_2 : \tau_2)$	by $\mathcal{D}_{[\text{For}]}$
3 :	$(\Gamma \vdash e_1 : \tau_1 \wedge E : e_1 \Downarrow v_1 \Rightarrow \Gamma \vdash v_1 : \tau_1)$ $\Rightarrow ((\Gamma \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 : \tau_2$ $\wedge\, E : \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2)$	by mp with 1.1 and 1.2
4 :	$\Gamma \vdash e_1 : \tau_1$	by $\mathcal{D}_{[\text{For}]}$
5 :	$E : e_1 \Downarrow v_1$	by inversion on $\mathcal{D}_{[\text{For}]}$
6 :	$\Gamma \vdash v_1 : \tau_1$	by IH_{GEN}
7 :	$(\Gamma \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 : \tau_2$ $\wedge\, E : \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2$	by mp with 1.3, 1.4, 1.5, and 1.6
8 :	$\Gamma \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 : \tau_2$	by $\mathcal{D}_{[\text{For}]}$
9 :	$E : \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow v_2$	by inversion on $\mathcal{D}_{[\text{For}]}$
10 :	$\Gamma \vdash v_2 : \tau_2$	by mp with 1.7, 1.8, and 1.9
$\therefore \mathcal{D}_{[\text{For}]}$ maintains IH_{GEN}		

Step 1 uses invokes the $\texttt{for}_{\text{AUX}}$ lemma, which is proved in here. The inductive hypothesis for the auxiliary judgments for the `for` expression is

$$\text{IH}_{\text{AUX}} = \begin{array}{l} (\Gamma \uplus \{\texttt{for}\, x : \tau_1\} \vdash e_2 : \tau_2\texttt{, } \texttt{C}_2) \\ \quad \Rightarrow ((\Gamma \vdash e_1 : \tau_1\texttt{, } \texttt{C}_1 \wedge E : e_1 \Downarrow v_1 \Rightarrow \Gamma \vdash v_1 : \tau_1\texttt{, } \texttt{C}_1) \\ \qquad \Rightarrow ((\Gamma \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 : \tau_2\texttt{, } \texttt{C}_1 \cup \texttt{C}_2 \\ \qquad\quad \wedge\, E : \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow v_2) \\ \qquad\quad \Rightarrow \Gamma \vdash v_2 : \tau_2\texttt{, } \texttt{C}_1 \cup \texttt{C}_2)) \end{array}$$

Proof 2. $\mathcal{D}_{[\text{ForU}]}$	
$1 : \Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1$	by assumption
$2 : E \vdash e_1 \Downarrow u$	by inversion and $\mathcal{E}_{[\text{For}]}$
$3 : \Gamma \vdash u : \tau_1,\ \mathtt{C}_1$	by IH_{GEN} on 2.1 and 2.2
$4 : u = u, \mathtt{()}$	by $\mathcal{S}_{\mathtt{NIL}}$, $\mathcal{F}_{\mathtt{NIL}}$, and 2.3
$5 : E \vdash e_2[u/x] \Downarrow v$	by inversion and $\mathcal{E}_{[\text{For}]}$
$6 : \Gamma \vdash e_2[u/x] : \tau,\ \mathtt{C}_1 \cup \mathtt{C}_2$	by sub. lemma and $\mathcal{D}_{[\text{For}]}$
$7 : \Gamma \vdash v : \tau,\ \mathtt{C}_1 \cup \mathtt{C}_2$	by 2.6
$8 : E \vdash \mathtt{for}\ x\ \mathtt{in}\ \mathtt{()}\ \mathtt{do}\ e_2 \Downarrow \mathtt{()}$	by $\mathcal{E}_{[\text{ForN}]}$
$9 : E \vdash \mathtt{for}\ x\ \mathtt{in}\ u\ \mathtt{do}\ e_2 \Downarrow v$	by 2.8 and $\mathcal{E}_{[\text{For}]}$
$\therefore \mathcal{D}_{[\text{ForU}]}$ maintains IH_{AUX}	

Proof 3. $\mathcal{D}_{[\text{ForN}]}$	
$1 : \Gamma \vdash \mathtt{()} : ()^0$	by def. of $()^0$
$2 : (\Gamma \vdash e_1 : ()^0 \wedge E \vdash e_1 \Downarrow \mathtt{()})$ $\Rightarrow \Gamma \vdash \mathtt{()} : ()^0$	by def. of "$\Rightarrow$"
$3 : E \vdash e_1 \Downarrow \mathtt{()}$ $\Rightarrow E \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow \mathtt{()}$	by $\mathcal{E}_{[\text{ForN}]}$
$4 : E \vdash e_1 \Downarrow \mathtt{()}$ $\Rightarrow E \vdash \mathtt{for}\ x\ \mathtt{in}\ \mathtt{()}\ \mathtt{do}\ e_2 \Downarrow \mathtt{()}$	by 3.3 and sub. lemma
$5 : \Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : ()^0$	by assumption
$6 : E \vdash e_1 \Downarrow \mathtt{()}$ $\Rightarrow \Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ \mathtt{()}\ \mathtt{do}\ e_2 : ()^0$	by 3.5 and sub. lemma
$7 : (\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ \mathtt{()}\ \mathtt{do}\ e_2 : ()^0$ $\wedge E \vdash \mathtt{for}\ x\ \mathtt{in}\ \mathtt{()}\ \mathtt{do}\ e_2 \Downarrow \mathtt{()})$ $\Rightarrow \Gamma \vdash \mathtt{()} : ()^0$	by 3.1, 3.4, and 3.6
$\therefore \mathcal{D}_{[\text{ForN}]}$ maintains IH_{AUX}	

Proof 4. $\mathcal{D}_{[\text{ForE}]}$	
$1 : \Gamma \vdash e_1 : \emptyset$	by assumption
$2 : \Gamma \vdash e_1 : \emptyset$ $\Rightarrow \neg(E \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v)$	because no rule exists for this case
$\therefore \mathcal{D}_{[\text{ForE}]}$ maintains IH_{AUX}	

Proof 5. $\mathcal{D}_{[\text{ForS}]}$	
$1 : (\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \wedge E : e_1 \Downarrow v_1$ $\Rightarrow \Gamma \vdash v_1 : \tau_1,\ \mathtt{C}_1)$ $\Rightarrow ((\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_1',$ $\mathtt{C}_1 \cup \mathtt{C}_1'$ $\wedge E : \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_1')$ $\Rightarrow \Gamma \vdash v_1' : \tau_1',\ \mathtt{C}_1 \cup \mathtt{C}_1{}')$	by $\mathcal{D}_{[\text{ForS}]}$
$2 : (\Gamma \vdash e_1 : \tau_2,\ \mathtt{C}_2 \wedge E : e_1 \Downarrow v_2$ $\Rightarrow \Gamma \vdash v_2 : \tau_2,\ \mathtt{C}_2)$ $\Rightarrow ((\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_2',$ $\mathtt{C}_2 \cup \mathtt{C}_2'$ $\wedge E : \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_2')$ $\Rightarrow \Gamma \vdash v_2' : \tau_2',\ \mathtt{C}_2 \cup \mathtt{C}_2{}')$	by $\mathcal{D}_{[\text{ForS}]}$
$3 : \Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1$	by assumption
$4 : E \vdash e_1 \Downarrow v_1$	by 5.3 and inversion
$5 : \Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \wedge E \vdash e_1 \Downarrow v_1$ $\Rightarrow \Gamma \vdash e_1 \vdash \tau_1,\ \mathtt{C}_1$	by 5.3, 5.4, and IH_{GEN}
$6 : \Gamma \vdash e_1' : \tau_1',\ \mathtt{C}_1 \cup \mathtt{C}_1' \wedge E \vdash e_1' \Downarrow v_1'$ $\Rightarrow \Gamma \vdash v_1' : \tau_1',\ \mathtt{C}_1 \cup \mathtt{C}_1'$	as in 5.3, 5.4, and 5.5
$7 : (\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ v_1\ \mathtt{do}\ e_2 : \tau_1',\ \mathtt{C}_1 \cup \mathtt{C}_1'$ $\wedge E : \mathtt{for}\ x\ \mathtt{in}\ v_1\ \mathtt{do}\ e_2 \Downarrow v_1')$ $\Rightarrow \Gamma \vdash v_1' : \tau_1',\ \mathtt{C}_1 \cup \mathtt{C}_1'$	by sub. lemma, 5.1, and 5.5

Proof 5. $\mathcal{D}_{[\text{ForS}]}$

8 :	$(\Gamma \vdash \texttt{for } x \texttt{ in } v_2 \texttt{ do } e_2 : \tau_2',\ \texttt{C}_2 \cup \texttt{C}_2'$ $\wedge E : \texttt{for } x \texttt{ in } v_2 \texttt{ do } e_2 \Downarrow v_2')$ $\Rightarrow \Gamma \vdash v_2' : \tau_2',\ \texttt{C}_2 \cup \texttt{C}_2'$	by 5.7
9 :	$v_1 = u_1, \ldots, u_m$	by list decomposition
10 :	$v_1, v_2 = u_1, \ldots, u_m$	by 5.9 and list composition
11 :	$E \vdash \texttt{for } x \texttt{ in } u_1,\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v' \Leftrightarrow$ $v' = u_1', v''$ where $E \uplus \{x \mapsto u_1\} \vdash e_2 \Downarrow u_1'$ and $E \vdash \texttt{for } x \texttt{ in } u_2,\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v''$	by $\mathcal{E}_{[\text{For}]}$
12 :	Suppose that for $i.\ 1 \leq i < m$ $E \vdash \texttt{for } x \texttt{ in } u_1,\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v' \Leftrightarrow$ $v' = u_1', \ldots, u_i', v''$ where $E \uplus \{x \mapsto u_1\} \vdash e_2 \Downarrow u_1'$, $\ldots$ $E \uplus \{x \mapsto u_i\} \vdash e_2 \Downarrow u_i'$ and $E \vdash \texttt{for } x \texttt{ in } u_{i+1},\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v''$	Inductive hypothesis for sequence proof
13 :	$E \vdash \texttt{for } x \texttt{ in } u_{i+1},\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v'' \Leftrightarrow$ $v'' = u_{i+1}', v'''$ where $E \uplus \{x \mapsto u_{i+1}\} \vdash e_2 \Downarrow u_{i+1}'$ and $E \vdash \texttt{for } x \texttt{ in } u_{i+2},\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v'''$	by 5.12 and $\mathcal{E}_{[\text{For}]}$
14 :	$E \vdash \texttt{for } x \texttt{ in } u_1,\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v' \Leftrightarrow$ $v' = u_1', \ldots, u_{i+1}', v'''$ where $E \uplus \{x \mapsto u_1\} \vdash e_2 \Downarrow u_1'$, $\ldots$ $E \uplus \{x \mapsto u_{i+1}\} \vdash e_2 \Downarrow u_{i+1}'$ and $E \vdash \texttt{for } x \texttt{ in } u_{i+2},\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v'''$	induction proved by 5.11, 5.12, and 5.13
15 :	$E \vdash \texttt{for } x \texttt{ in } u_1,\ldots,u_m,v_2 \texttt{ do } e_2 \Downarrow v' \Leftrightarrow$ $v' = u_1', \ldots, u_m', v'''$ where $E \uplus \{x \mapsto u_1\} \vdash e_2 \Downarrow u_1'$, $\ldots$ $E \uplus \{x \mapsto u_m\} \vdash e_2 \Downarrow u_m'$ and $E \vdash \texttt{for } x \texttt{ in } v_2 \texttt{ do } e_2 \Downarrow v'''$	by 5.14
16 :	$v''' = v_2' \wedge \Gamma \vdash v_2' : \tau_2',\ \texttt{C}_2 \cup \texttt{C}_2'$	by 5.8 and 5.15
17 :	$u_1', \ldots, u_m' = v_1' \wedge \Gamma \vdash v_1' : \tau_1',\ \texttt{C}_1 \cup \texttt{C}_1'$	by 5.7, 5.14, and $\mathcal{E}_{[\text{For}]}$
18 :	$v' = v_1', v_2'$	by 5.15, 5.16, and 5.17
19 :	$\Gamma \vdash v' : (\tau_1', \tau_2')^{\texttt{r}},\ \texttt{C}_1 \cup \texttt{C}_1'$ $\cup\ \texttt{C}_2 \cup \texttt{C}_2' \cup \{\texttt{n} = \texttt{m}_1 + \texttt{m}_2\}$	by $\mathcal{D}_{[\text{Seq}]}$, 5.16, 5.17, and 5.18
	$\therefore \mathcal{D}_{[\text{ForS}]}$ maintains IH_{AUX}	

Proof 6. $\mathcal{D}_{[\text{ForC}]}$

1 :	$\vdash d : \tau_1,\ \texttt{C} \Rightarrow \vdash d : (\tau_1 \mid \tau_2)^{\texttt{n}},\ \texttt{C} \cup \texttt{C}_2$	by $\mathcal{F}_{\text{CHC1}}$
2 :	$\tau_1 <: (\tau_1 \mid \tau_2)^{\texttt{n}}$	by def. of subsumption
3 :	$\Gamma \vdash e_1 : \tau_1 \wedge E \vdash e_1 \Downarrow v_1$	by assumption
4 :	$(\Gamma \vdash e_1 : \tau_1,\ \texttt{C}_1 \wedge E \vdash e_1 \Downarrow v_1)$ $\Rightarrow \Gamma \vdash v_1 : \tau_1,\ \texttt{C}_1$	by IH_{AUX}
5 :	$(\Gamma \vdash e_1 : (\tau_1 \mid \tau_1')^{\texttt{n}},\ \texttt{C}_1 \cup \texttt{C}_1'$ $\wedge E \vdash e_1 \Downarrow v_1)$ $\Rightarrow \Gamma \vdash v_1 : (\tau_1 \mid \tau_1')^{\texttt{n}},\ \texttt{C}_1 \cup \texttt{C}_1'$	by 6.2 and 6.4
6 :	$\Gamma \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 : \tau_2,$ $\texttt{C}_1 \cup \texttt{C}_2$ $\wedge E \vdash \texttt{for } x \texttt{ in } e_1 \texttt{ do } e_2 \Downarrow v_2$	by $\mathcal{D}_{[\text{For}]}$, inversion and $\mathcal{E}_{[\text{For}]}$
7 :	$\Gamma \vdash \texttt{for } x \texttt{ in } v_1 \texttt{ do } e_2 : \tau_2,$ $\texttt{C}_1 \cup \texttt{C}_2$ $\wedge E \vdash \texttt{for } x \texttt{ in } v_1 \texttt{ do } e_2 \Downarrow v_2$	by 6.5 and 6.6
8 :	$\Gamma \vdash v_2 : \tau_2,\ \texttt{C}_1 \cup \texttt{C}_2$	by 6.7 and IH_{AUX}
9 :	$\Gamma \vdash \texttt{for } x \texttt{ in } v_1 \texttt{ do } e_2 : (\tau_2 \mid \tau_2')^{\texttt{m}},$ $\texttt{C}_1 \cup \texttt{C}_1' \cup \texttt{C}_2 \cup \texttt{C}_2'$ $\wedge E \vdash \texttt{for } x \texttt{ in } v_1 \texttt{ do } e_2 \Downarrow v_2 \wedge$ $\Gamma \vdash v_2 : (\tau_2 \mid \tau_2')^{\texttt{m}},\ \texttt{C}_1 \cup \texttt{C}_1' \cup \texttt{C}_2 \cup \texttt{C}_2'$	by 6.2, 6.7, and 6.8

Proof 6. $\mathcal{D}_{[\textsc{ForC}]}$		
10 :	The same reasoning applies to $\Gamma \uplus \{\mathtt{for}\ x : \tau_2\} \vdash e_2 : \tau_2'$	by syntactic substitution
	$\therefore \mathcal{D}_{[\textsc{ForC}]}$ maintains IH_{AUX}	

Proof 7. $\mathcal{D}_{[\textsc{ForP}]}$		
1 :	Except that $<\tau_1,\tau_2>^{\mathtt{n}} \neq <\tau_2,\tau_1>^{\mathtt{n}}$, the semantics of $<\tau_1,\tau_2>^{\mathtt{n}}$ is equivalent to the semantics of $(\tau_1\|\tau_2)^{\mathtt{n}}$.	by rule set
2 :	$(\Gamma \uplus \{\mathtt{for}\ x : <\tau_1,\tau_2>^{\mathtt{n}}\} \vdash e_2 : <\tau_1',\tau_2'>^{\mathtt{n}'},$ $\mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = \mathtt{b} \times \mathtt{r}_1 + \mathtt{notb} \times \mathtt{r}_2\}$ $\cup\ \mathtt{C}_1' \cup \mathtt{C}_2' \cup \{\mathtt{n}' = \mathtt{b} \times \mathtt{m}_1 + \mathtt{notb} \times \mathtt{m}_2\})$ $\Rightarrow ((\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \wedge E \vdash e_1 \Downarrow v_1$ $\Rightarrow \Gamma \vdash v_1 : \tau_1,\ \mathtt{C}_1)$ $\Rightarrow ((\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_1',$ $\mathtt{C}_1 \cup \mathtt{C}_1'$ $\wedge E : \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_1',\ \mathtt{C}_1 \cup \mathtt{C}_1'))$	by 6.9, 6.10, 7.1, and IH_{AUX} on τ_1
3 :	$(\Gamma \uplus \{\mathtt{for}\ x : <\tau_1,\tau_2>^{\mathtt{n}}\} \vdash e_2 : <\tau_1',\tau_2'>^{\mathtt{n}'},$ $\mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = \mathtt{b} \times \mathtt{r}_1 + \mathtt{notb} \times \mathtt{r}_2\}$ $\cup\ \mathtt{C}_1' \cup \mathtt{C}_2' \cup \{\mathtt{n}' = \mathtt{b} \times \mathtt{m}_1 + \mathtt{notb} \times \mathtt{m}_2\})$ $\Rightarrow ((\Gamma \vdash e_1 : \tau_2,\ \mathtt{C}_2 \wedge E : e_1 \Downarrow v_1$ $\Rightarrow \Gamma \vdash v_1 : \tau_2,\ \mathtt{C}_2)$ $\Rightarrow ((\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_2',$ $\mathtt{C}_2 \cup \mathtt{C}_2'$ $\wedge E : \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2',\ \mathtt{C}_2 \cup \mathtt{C}_2'))$	as in 7.2 on τ_2
4 :	for every $E \vdash e_1 \Downarrow v_1\ \wedge\ \Gamma \vdash v_1 : \tau_i$ there is a $E \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_2$ such that $\Gamma \vdash v_2 : \tau_i'$	by 7.2 and 7.3
5 :	$(\Gamma \uplus \{\mathtt{for}\ x : <\tau_1,\tau_2>^{\mathtt{n}}\} \vdash e_2 : <\tau_1',\tau_2'>^{\mathtt{n}'},$ $\mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = \mathtt{b} \times \mathtt{m}_1 + \mathtt{notb} \times \mathtt{m}_2\}\ \ (= \mathtt{C})$ $\cup\ \{\mathtt{n}' = \mathtt{b} \times \mathtt{m}_1' + \mathtt{notb} \times \mathtt{m}_2'\}\ \ (= \mathtt{C}'))$ $\Rightarrow ((\Gamma \vdash e_1 : <\tau_1,\tau_2>^{\mathtt{n}} \wedge E : e_1 \Downarrow v_1 \Rightarrow$ $\Gamma \vdash v_1 : <\tau_2,\tau_2>^{\mathtt{n}},\ \mathtt{C}_1 \cup \mathtt{C}_2 \cup$ $\{\mathtt{n} = \mathtt{b} \times \mathtt{r}_1 = \mathtt{notb} \times \mathtt{r}_2\})$ $\Rightarrow ((\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_1',\ \mathtt{C}_1'$ $\wedge E : \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_1',\ \mathtt{C}_1'))$	by 7.4, 7.1, and post-processing predicates
	$\therefore \mathcal{D}_{[\textsc{ForP}]}$ maintains IH_{AUX}	

Proof 8. $\mathcal{D}_{[\textsc{ForR}]}$		
1 :	$\forall \tau.\ \vdash d : (\tau, \tau{*}^{\mathtt{n}})^{\mathtt{r}}$ $\Rightarrow (\exists d_1, d_2.\ d_1, d_2 = d \wedge\ \vdash d_1 : \tau \wedge\ \vdash d_2 : \tau{*}^{\mathtt{n}})$	by $\mathcal{F}_{\text{SEQ}}$
2 :	$\vdash d_1 : \tau \wedge\ \vdash d_2 : \tau{*}^{\mathtt{n}}\ \Rightarrow \vdash d_1, d_2 : \tau{*}^{\mathtt{r}}$	by $\mathcal{F}_{\text{REP}}$
3 :	$d_1, d_2 = d \wedge\ \vdash d_1, d_2 : \tau{*}^{\mathtt{r}}\ \Rightarrow \vdash d : \tau{*}^{\mathtt{r}}$	by substitution
4 :	$\vdash d : \tau, \tau{*}^{\mathtt{n}} \Rightarrow \vdash d : \tau{*}^{\mathtt{r}}$	by transitivity: 8.1 $\Rightarrow$ 8.2 $\Rightarrow$ 8.3
5 :	$\tau, \tau{*}^{\mathtt{n}} <: \tau{*}^{\mathtt{r}}$	by 8.4 and def. of subsumption
6 :	$\vdash d : \mathtt{()} \Rightarrow \vdash d : \tau{*}^{\mathtt{n}}$	by $\mathcal{F}_{\text{REP()}}$
7 :	$() <: \tau{*}^{\mathtt{r}}$	by 8.6 and def. of subsumption
8 :	$\vdash d : \tau{*}^{\mathtt{r}}\ \wedge \neg(\vdash d : ())$ $\Rightarrow\ (\exists d_1, d_2.\ d_1, d_2 = d \wedge\ \vdash d_1 : \tau \wedge\ \vdash d_2 : \tau{*}^{\mathtt{r}})$	by uniqueness of $\mathcal{F}_{\text{REP}}$ and $\mathcal{F}_{\text{REP()}}$
9 :	$()\|\tau, \tau{*}^{\mathtt{n}}\ <:\ \tau{*}^{\mathtt{r}}$	by $\mathcal{S}_{\text{choice}}$
10 :	$\tau{*}^{\mathtt{r}}\ <:\ ()\|\tau, \tau{*}^{\mathtt{n}}$	by 8.8 and def. of subsumption
11 :	$\tau{*}^{\mathtt{r}}\ =\ (\tau, \tau{*}^{\mathtt{n}}\|())$	by $\mathcal{S}_{\text{eq}}$
12 :	$\Gamma \vdash e_1 : \tau{*}^{\mathtt{n}}$	by assumption
13 :	$\Gamma \vdash e : ()\ \vee\ \Gamma \vdash e : \tau, \tau{*}^{\mathtt{n}}$	by 8.12 and 8.11
14 :	$\Gamma \vdash e : ()\ \Rightarrow\ E \vdash e \Downarrow \mathtt{()}$	by assumption and inversion on e
15 :	$\Gamma \vdash e : () \wedge E \vdash e \Downarrow \mathtt{()}\ \Rightarrow\ \Gamma \vdash \mathtt{()} : ()$	by def. of ()

Proof 8. $\mathcal{D}_{[\text{ForR}]}$	
$16: \Gamma \vdash e : () \wedge E \vdash e \Downarrow \texttt{()} \;\Rightarrow\; \Gamma \vdash \texttt{()} : \tau *^{\mathrm{n}}$	by 8.7
$17: E \vdash e \Downarrow \texttt{()} \;\Rightarrow\; E \vdash \texttt{for}\ x\ \texttt{in}\ e\ \texttt{do}\ e_2 \Downarrow \texttt{()}$	by $\mathcal{E}_{[\text{ForN}]}$
$18: E \vdash e \Downarrow \texttt{()} \;\Rightarrow\; \Gamma \vdash \texttt{for}\ x\ \texttt{in}\ e\ \texttt{do}\ e_2 : ()$	by $\mathcal{D}_{[\text{ForN}]}$
19: $E \vdash \texttt{for}\ x\ \texttt{in}\ \texttt{()}\ \texttt{do}\ e_2 \Downarrow \texttt{()}$ $\wedge\, \Gamma \vdash \texttt{for}\ x\ \texttt{in}\ \texttt{()}\ \texttt{do}\ e_2 : ()$ $\Rightarrow \Gamma \vdash \texttt{()} : ()$	by 8.17, 8.18 and def. of $\texttt{()}$
20: $(\Gamma \vdash e_1 : () \wedge E : e_1 \Downarrow \texttt{()} \Rightarrow \Gamma \vdash \texttt{()} : ())$ $\Rightarrow ((\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ \texttt{()}\ \texttt{do}\ e_2 : \tau *^{\mathrm{n}}$ $\wedge\, E \vdash \texttt{for}\ x\ \texttt{in}\ \texttt{()}\ \texttt{do}\ e_2 \Downarrow \texttt{()})$ $\Rightarrow \Gamma \vdash \texttt{()} : \tau *^{\mathrm{n}})$	by 8.7 and 8.19
21: $(\Gamma \vdash e_1 : \tau \wedge E : e_1 \Downarrow v_1 \Rightarrow \Gamma \vdash v_1 : \tau)$ $\Rightarrow ((\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 : \tau_2$ $\wedge\, E : \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2)$	by $\mathcal{D}_{[\text{ForS}]}$
$22: \tau <: \tau *^{\mathrm{r}}$	by 8.5 and 8.7
23: $(\Gamma \vdash e_1 : \tau *^{\mathrm{n}} \wedge E : e_1 \Downarrow v_1 \Rightarrow \Gamma \vdash v_1 : \tau *^{\mathrm{n}})$ $\Rightarrow ((\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 : \tau_2 *^{\mathrm{r}}$ $\wedge\, E : \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2 *^{\mathrm{r}})$	by 8.21 and 8.22
24: Suppose that $\forall v = v_m, \ldots, v_1$ where $\Gamma \vdash v_i : \tau$ $(\Gamma \vdash e_1 : \tau *^{\mathrm{n}} \wedge E : e_1 \Downarrow v \Rightarrow \Gamma \vdash v : \tau *^{\mathrm{n}})$ $\Rightarrow ((\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 : \tau_2 *^{\mathrm{r}}$ $\wedge\, E : \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2 *^{\mathrm{r}})$	inductive hypothesis for repetition proof with 8.20 and 8.23 as base cases
25: let $v' = v_{m+1}\,,v$ where $\Gamma \vdash v_{m+1} : \tau \;\wedge\; \Gamma \vdash v : \tau *^{\mathrm{n}}$	for inductive step
$26: \Gamma \vdash v' : \tau *^{\mathrm{n}'}$	by 8.25 and $\mathcal{F}_{\text{REP}}$
27: $\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ v_{m+1}\ \texttt{do}\ e_2 : \tau_2$ $\wedge\, E \vdash \texttt{for}\ x\ \texttt{in}\ v_{m+1}\ \texttt{do}\ e_2 \Downarrow v'_{m+1}$ $\Rightarrow \Gamma \vdash v'_{m+1} : \tau_2$	by 8.23
28: $\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ v\ \texttt{do}\ e_2 : \tau_2 *^{\mathrm{r}}$ $\wedge\, E \vdash \texttt{for}\ x\ \texttt{in}\ v\ \texttt{do}\ e_2 \Downarrow v''$ $\Rightarrow \Gamma \vdash v'' : \tau_2 *^{\mathrm{r}}$	by 8.24
29: $\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ v'\ \texttt{do}\ e_2 : (\tau_2, \tau_2 *^{\mathrm{r}})^{\mathrm{r}'}$ $\wedge\, E \vdash \texttt{for}\ x\ \texttt{in}\ v'\ \texttt{do}\ e_2 \Downarrow v_{m+1}\,, v''$ $\Rightarrow \Gamma \vdash v_{m+1}\,, v'' : (\tau_2, \tau_2 *^{\mathrm{r}})^{\mathrm{r}'}$	by $\mathcal{D}_{[\text{ForS}]}$
30: $\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ v'\ \texttt{do}\ e_2 : \tau_2 *^{\mathrm{r}'}$ $\wedge\, E \vdash \texttt{for}\ x\ \texttt{in}\ v'\ \texttt{do}\ e_2 \Downarrow v_{m+1}\,, v''$ $\Rightarrow \Gamma \vdash v_{m+1}\,, v'' : \tau_2 *^{\mathrm{r}'}$	by $\mathcal{F}_{\text{REP}}$
$\therefore\ \mathcal{D}_{[\text{ForR}]}$ maintains IH_{AUX}	

$$
\text{IH}_{\text{AUX}\Pi} = \begin{array}{l}
(\Gamma \uplus \{\texttt{for}\ x : \tau_1\} \vdash e_2 : \tau_2,\ \texttt{C}_2 \,\cup\, \{\pi\}) \\
\quad \Rightarrow ((\Gamma \vdash e_1 : \tau_1,\ \texttt{C}_1 \wedge E \vdash e_1 \Downarrow v_1 \Rightarrow \Gamma \vdash v_1 : \tau_1,\ \texttt{C}_1) \\
\qquad \Rightarrow ((\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 : \tau_2,\ \texttt{C}_1 \,\cup\, \texttt{C}_2 \,\cup\, \{\pi\} \\
\qquad\quad \wedge\, E \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 \Downarrow v_2) \\
\qquad\quad \Rightarrow \Gamma \vdash v_2 : \tau_2,\ \texttt{C}_1 \,\cup\, \texttt{C}_2 \,\cup\, \{\pi\}))
\end{array}
$$

Proof 9. $\mathcal{D}_{[\text{For}\Pi]}$	
1: $(\Gamma \uplus \{\texttt{for}\ x : \tau_1\} \vdash e_2 : \tau_2,\ \{\pi\} \,\cup\, \texttt{C}_2)$ $\Rightarrow ((\Gamma \vdash e_1 : \tau_1,\ \texttt{C}_1 \wedge E : e_1 \Downarrow v_1$ $\Rightarrow \Gamma \vdash v_1 : \tau_1,\ \texttt{C}_1)$ $\Rightarrow ((\Gamma \vdash \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 : \tau_2,$ $\{\pi\} \,\cup\, \texttt{C}_1 \,\cup\, \texttt{C}_2$ $\wedge\, E : \texttt{for}\ x\ \texttt{in}\ e_1\ \texttt{do}\ e_2 \Downarrow v_2)$ $\Rightarrow \Gamma \vdash v_2 : \tau_2,\ \{\pi\} \,\cup\, \texttt{C}_1 \,\cup\, \texttt{C}_2))$	by the $\text{IH}_{\text{AUX}\Pi}$ lemma
$2: (\Gamma \uplus \{\texttt{for}\ x : \tau_1\} \vdash e_2 : \tau_2,\ \{\pi\} \,\cup\, \texttt{C}_2)$	by $\mathcal{D}_{[\text{For}\Pi]}$

Proof 9. $\mathcal{D}_{[\text{FOR}\Pi]}$	
3 : $\Rightarrow ((\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1 \land E : e_1 \Downarrow v_1 \Rightarrow \Gamma \vdash v_1 : \tau_1,\ \mathtt{C}_1) \Rightarrow ((\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_2,\ \{\pi\} \cup \mathtt{C}_1 \cup \mathtt{C}_2 \land E : \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_2) \Rightarrow \Gamma \vdash v_2 : \tau_2,\ \{\pi\} \cup \mathtt{C}_1 \cup \mathtt{C}_2))$	by mp with 9.1 and 9.2
4 : $\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1$	by $\mathcal{D}_{[\text{FOR}\Pi]}$
5 : $E : e_1 \Downarrow v_1$	by inversion on $\mathcal{D}_{[\text{FOR}\Pi]}$
6 : $\Gamma \vdash v_1 : \tau_1,\ \mathtt{C}_1$	by IH_{GEN}
7 : $(\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : \tau_2,\ \{\pi\} \cup \mathtt{C}_1 \cup \mathtt{C}_2 \land E : \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v_2) \Rightarrow \Gamma \vdash v_2 : \tau_2,\ \{\pi\} \cup \mathtt{C}_1 \cup \mathtt{C}_2$	by mp with 9.3, 9.4, 9.5, and 9.6
8 : $\mathtt{for}.label = start \Rightarrow$ no output is produced before $e_2{}'$ that iterates over $d \in s$	by pred1 and def1
9 : $(z^{\mathtt{m}})^{*\mathtt{n}},\ \mathtt{C}_2 \cup \{\pi\}$ came from $\mathcal{D}_{[\text{FOR}\Pi\text{U}]}$	by uniqueness of $\mathcal{D}_{[\text{FOR}\Pi\text{U}]}$ and pred2
10 : $E \vdash e_2' \Downarrow v' \Rightarrow \Gamma \vdash v' : z^{\mathtt{m}},\ \mathtt{C}_2$	by 9.8, 9.9, and $\mathcal{D}_{[\text{FOR}\Pi 1]}$
11 : $e_2' \Downarrow \ldots$ once for each $d \in s$	by pred1
12 : $\Gamma_\pi(\pi) = \|s\|$	by def1
13 : $v = v_1', \ldots, v'_{\Gamma_\pi(\pi)}$	by 9.11 and 9.12
14 : $\Gamma \vdash v : (z^{\mathtt{m}})^{*\mathtt{n}},\ \mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\mathtt{n} = \mathtt{m} \times \Gamma_\pi(\pi)\}$	by 9.10 and 9.13
$\therefore \mathcal{D}_{[\text{FOR}\Pi]}$ maintains IH_{GEN}	

Proof 10. $\mathcal{D}_{[\text{FORU}]}$	
1 : $\Gamma \vdash e_1 : \tau_1,\ \mathtt{C}_1$	by assumption
2 : $E \vdash e_1 \Downarrow u$	by inversion and $\mathcal{E}_{[\text{FOR}]}$
3 : $\Gamma \vdash u : \tau_1,\ \mathtt{C}_1$	by IH_{GEN} on 10.1 and 10.2
4 : $u = u, \mathtt{()}$	by $\mathcal{S}_{\mathtt{NIL}}$, $\mathcal{F}_{\mathtt{NIL}}$, and 10.3
5 : $E \vdash e_2[u/x] \Downarrow v$	by inversion and $\mathcal{E}_{[\text{FOR}]}$
6 : $\Gamma \vdash e_2[u/x] : \tau,\ \mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\pi\}$	by sub. lemma and $\mathcal{D}_{[\text{FORU}]}$
7 : $\Gamma \vdash v : \tau,\ \mathtt{C}_1 \cup \mathtt{C}_2 \cup \{\pi\}$	by 10.6
8 : $E \vdash \mathtt{for}\ x\ \mathtt{in}\ \mathtt{()}\ \mathtt{do}\ e_2 \Downarrow \mathtt{()}$	by $\mathcal{E}_{[\text{FORN}]}$
9 : $E \vdash \mathtt{for}\ x\ \mathtt{in}\ u\ \mathtt{do}\ e_2 \Downarrow v$	by 10.8 and $\mathcal{E}_{[\text{FOR}]}$
$\therefore \mathcal{D}_{[\text{FORU}]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 11. $\mathcal{D}_{[\text{FOR}\Pi\text{U}]}$	
1 : $x.label = \pi \Rightarrow \forall x'.label = \pi' \in e_2\ \exists \mathtt{for}\ x'\ \mathtt{in}\ e_1{}'\ \mathtt{do}\ e_2{}' \in e_2.\ x' \in e_2' \land \mathtt{for}.label = start$	by preproc. pred1
2 : $\mathtt{for}.label = start \Rightarrow \{\pi'\} \notin \mathtt{C}$	by $\mathcal{D}_{[\text{FOR}\Pi\text{U}]}$
3 : $\{\pi'\} \notin \mathtt{C}$	by 11.1 and 11.2
4 : $\Gamma \vdash e_1 : u^{\mathtt{1}}$	by assumption from $\mathcal{D}_{[\text{FOR}\Pi\text{U}]}$
5 : $E \vdash e_1 \Downarrow v_u$	by inversion from $\mathcal{D}_{[\text{FOR}\Pi\text{U}]}$
6 : $\Gamma \vdash v_u : u^{\mathtt{1}}$	by IH_{GEN}
7 : $E \uplus \{x \mapsto v_u\} \vdash e_2 \Downarrow v'$	by inversion
8 : $\Gamma \uplus \{x : u^{\mathtt{1}}\} \vdash v' : z^{\mathtt{m}},\ \mathtt{C}$	by IH_{GEN} and sub. lemma
9 : $z^{\mathtt{m}} <: (z^{\mathtt{m}}) *^{\mathtt{n}}$	by $\mathcal{S}_{\text{UREP}}$
10 : $E \vdash \mathtt{for}\ x\ \mathtt{in}\ \mathtt{()}\ \mathtt{do}\ e_2 \Downarrow \mathtt{()}$	by $\mathcal{E}_{[\text{FORN}]}$
11 : $E \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 \Downarrow v'$	by $\mathcal{E}_{[\text{FOR}]}$, 11.5, 11.7, and 11.10
12 : $\Gamma \vdash \mathtt{for}\ x\ \mathtt{in}\ e_1\ \mathtt{do}\ e_2 : z^{\mathtt{m}},\ \mathtt{C}$	by 11.6, 11.8, and $\mathcal{D}_{[\text{FOR}\Pi\text{U}]}$
13 : $\Gamma \vdash v' : z^{\mathtt{m}},\ \mathtt{C}$	by 11.8, 11.11, and 11.12
14 : $\Gamma \vdash v' : (z^{\mathtt{m}})^{*\mathtt{n}},\ \mathtt{C}$	by 11.9 and 11.13
15 : $\Gamma \vdash v' : (z^{\mathtt{m}})^{*\mathtt{n}},\ \mathtt{C} \cup \{\pi\}$	by 11.14 and because $x.label = \pi$
$\therefore \mathcal{D}_{[\text{FOR}\Pi\text{U}]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 12. $\mathcal{D}_{[\textsc{For}\Pi 1]}$	
1 : $\{\pi\}$ was added to both $\mathtt{C}$'s from $\mathcal{D}_{[\textsc{For}\Pi\text{U}]}$ on the same expression	by pred1 and uniqueness of $\mathcal{D}_{[\textsc{For}\Pi\text{U}]}$ for $x.label = \pi$
2 : $\tau' = (z^{\mathtt{m}})^{*\mathtt{n}}$	by 12.1 and $\mathcal{D}_{[\textsc{For}\Pi\text{U}]}$
3 : If $\tau = (\tau_{1a}, \tau_{1b})^{\mathtt{n}}$:	
4 : $((z^{\mathtt{m}})^{*\mathtt{n}}, (z^{\mathtt{m}})^{*\mathtt{n}})^{*\mathtt{n}'} = (z^{\mathtt{m}})^{*\mathtt{n}'}$	by $\mathcal{S}_{\text{SREP}}$
5 : $(z^{\mathtt{m}})^{*\mathtt{n}'} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by unbound variable renaming
6 : $(\tau', \tau')^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 12.2, 12.5, and 12.4
7 : If $\tau = (\tau_{1a} \vert \tau_{1b})^{\mathtt{n}}$:	
8 : $(z^{\mathtt{m}})^{*\mathtt{n}} \vert (z^{\mathtt{m}})^{*\mathtt{n}} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by $\mathcal{S}_{\text{SCHC}}$
9 : $(\tau' \vert \tau')^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 12.2 and 12.8
10 : If $\tau = <\tau_{1a}, \tau_{1b}>^{\mathtt{n}}$:	
11 : $<(z^{\mathtt{m}})^{*\mathtt{n}}, (z^{\mathtt{m}})^{*\mathtt{n}}> = (z^{\mathtt{m}})^{*\mathtt{n}}$	by $\mathcal{S}_{\text{SPR}}$
12 : $<\tau', \tau'>^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 12.2 and 12.11
$\therefore \mathcal{D}_{[\textsc{For}\Pi 1]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 13. $\mathcal{D}_{[\textsc{For}\Pi 2]}$	
1 : $\{\pi\}$ was added to both $\mathtt{C}$'s from $\mathcal{D}_{[\textsc{For}\Pi\text{U}]}$ on the same expression	by pred1 and uniqueness of $\mathcal{D}_{[\textsc{For}\Pi\text{U}]}$ for $x.label = \pi$
2 : $\tau' = (z^{\mathtt{m}})^{*\mathtt{n}}$	by 13.1 and $\mathcal{D}_{[\textsc{For}\Pi\text{U}]}$
3 : If $\tau = (\tau_{1a}, \mathtt{()}^{0})^{\mathtt{n}}$:	
4 : $((z^{\mathtt{m}})^{*\mathtt{n}}, \mathtt{()}^{0})^{*\mathtt{n}} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by $\mathcal{S}_{\text{SNIL}}$
5 : $(\tau', \mathtt{()}^{0})^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 13.2 and 13.4
6 : If $\tau = (\tau_{1a} \vert \mathtt{()}^{0})^{\mathtt{n}'}$:	
7 : $((z^{\mathtt{m}})^{*\mathtt{n}} \vert \mathtt{()}^{0})^{\mathtt{n}'} = (z^{\mathtt{m}})^{*\mathtt{n}'}$	by $\mathcal{S}_{\text{SCHC}}$
8 : $(z^{\mathtt{m}})^{*\mathtt{n}'} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by unbound variable renaming
9 : $(\tau' \vert \mathtt{()}^{0})^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 13.2, 13.8, and 13.7
10 : If $\tau = <\tau_{1a}, \mathtt{()}^{0}>^{\mathtt{n}'}$:	
11 : $<(z^{\mathtt{m}})^{*\mathtt{n}}, \mathtt{()}^{0}>^{\mathtt{n}'} = (z^{\mathtt{m}})^{*\mathtt{n}'}$	by $\mathcal{S}_{\text{SPR}}$
12 : $(z^{\mathtt{m}})^{*\mathtt{n}'} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by unbound variable renaming
13 : $<\tau', \mathtt{()}^{0}>^{\mathtt{n}'} = \tau'$, $\mathtt{C}$	by 13.2, 13.12, and 13.11
$\therefore \mathcal{D}_{[\textsc{For}\Pi 2]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 14. $\mathcal{D}_{[\textsc{For}\Pi 3]}$	
1 : This proof exactly follows Proof 13	by type symmetry
$\therefore \mathcal{D}_{[\textsc{For}\Pi 3]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 15. $\mathcal{D}_{[\textsc{For}\Pi 4]}$	
1 : $\tau' = (z^{\mathtt{m}})^{*\mathtt{n}}$	by 15.0 and $\mathcal{D}_{[\textsc{For}\Pi\text{U}]}$
2 : If $\tau = (\tau_{1a}, \emptyset)^{\mathtt{n}}$:	
3 : $((z^{\mathtt{m}})^{*\mathtt{n}}, \emptyset)^{*\mathtt{n}} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by $\mathcal{S}_{\text{SEMT}}$
4 : $(\tau', \emptyset)^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 15.1 and 15.3
5 : If $\tau = (\tau_{1a} \vert \emptyset)^{\mathtt{n}}$:	
6 : $((z^{\mathtt{m}})^{*\mathtt{n}} \vert \emptyset)^{\mathtt{n}} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by $\mathcal{S}_{\text{SCHC}}$
7 : $(\tau' \vert \emptyset)^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 15.1 and 15.6
8 : If $\tau = <\tau_{1a}, \emptyset>^{\mathtt{n}}$:	
9 : $<(z^{\mathtt{m}})^{*\mathtt{n}}, \emptyset>^{\mathtt{n}} = (z^{\mathtt{m}})^{*\mathtt{n}}$	by $\mathcal{S}_{\text{SPR}}$
10 : $<\tau', \emptyset>^{\mathtt{n}} = \tau'$, $\mathtt{C}$	by 15.1 and 15.9
$\therefore \mathcal{D}_{[\textsc{For}\Pi 4]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 16. $\mathcal{D}_{[\textsc{For}\Pi 5]}$	
1 : This proof exactly follows Proof 15	by type symmetry
$\therefore \mathcal{D}_{[\textsc{For}\Pi 5]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 17. $\mathcal{D}_{[\text{For}\Pi\text{R}]}$	
$1: \tau' = (\tau')^{*\mathtt{r}'} \Rightarrow$ $\mathcal{D}_{[\text{For}\Pi\text{R}]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	by proof of $\mathcal{D}_{[\text{For}\Pi\text{R}]}$
$2: ()^0 = (()^0)^{*\mathtt{r}'}$	by $\mathcal{S}_{\text{NREP}}$
$3: \tau' = ()^0 \Rightarrow \tau' = (\tau')^{*\mathtt{r}'}$	by 17.2
$4: \tau' \neq ()^0 \Rightarrow$ $\tau' = (\tau'')^{*\mathtt{n}} \wedge \mathtt{n} \notin \mathtt{C}$	by $\mathcal{D}_{[\text{For}\Pi\text{U}]}$
$5: (\tau'')^{*\mathtt{n}} = ((\tau'')^{*\mathtt{n}})^{*\mathtt{r}'}$	by $\mathcal{S}_{\text{RNST}}$
$6: \tau' = (\tau')^{*\mathtt{r}'}$	by 17.3 and 17.5
$\therefore \mathcal{D}_{[\text{For}\Pi\text{R}]}$ maintains $\text{IH}_{\text{AUX}\Pi}$	

Proof 18. $\mathcal{D}_{[\text{Int}]}$	
$1: c_{\text{int}} \in \mathtt{Int}$	by def. of types and of $\mathtt{Int}$
$\therefore \mathcal{D}_{[\text{Int}]}$ maintains IH_{GEN}	

Proof 19. $\mathcal{D}_{[\text{Var}]}$	
$1: \{v/x\} \in E \Leftrightarrow (\Gamma \vdash v : \tau \Leftrightarrow (x : \tau) \in \Gamma)$	by def. of E
$2: \{v/x\} \in E \Leftrightarrow E \vdash e_{v/x} \Downarrow v'$	by def. of E
$3: E \vdash x_{v/x} \Downarrow v$	by 19.2
$4: \Gamma \vdash v : \tau$	by 19.1
$\therefore \mathcal{D}_{[\text{Var}]}$ maintains IH_{GEN}	

Proof 20. $\mathcal{D}_{[\text{Elt}]}$	
$1: E \vdash e \Downarrow v$	by inversion and $\mathcal{E}_{[\text{Elt}]}$
$2: \Gamma \vdash v : \tau$	by 20.1 and IH_{GEN}
$3: \Gamma \vdash a[v] : a[\tau]$	by 20.2
$\therefore \mathcal{D}_{[\text{Elt}]}$ maintains IH_{GEN}	

Proof 21. $\mathcal{D}_{[\text{Plus}]}$	
$1: E \vdash e_1 \Downarrow v_1$	by inversion and $\mathcal{E}_{[\text{Plus}]}$
$2: \Gamma \vdash v_1 : \mathtt{Int}^1,\ \mathtt{C}_1$	by 21.1 and IH_{GEN}
$3: E \vdash e_2 \Downarrow v_2$	by inversion and $\mathcal{E}_{[\text{Plus}]}$
$4: \Gamma \vdash v_2 : \mathtt{Int}^1,\ \mathtt{C}_2$	by 21.3 and IH_{GEN}
$5: \Gamma \vdash v_1 + v_2 : \mathtt{Int}^1$	by 21.2, 21.4, and def. of apply
$\therefore \mathcal{D}_{[\text{Plus}]}$ maintains IH_{GEN}	

Proof 22. $\mathcal{D}_{[\text{Eq}]}$	
$1: E \vdash e_1 \Downarrow u_1$	by inversion and $\mathcal{E}_{[\text{Eq}]}$
$2: \Gamma \vdash u_1 : u^1,\ \mathtt{C}_1$	by 22.1 and IH_{GEN}
$3: E \vdash e_2 \Downarrow u_2$	by inversion and $\mathcal{E}_{[\text{Eq}]}$
$4: \Gamma \vdash u_2 : u^1,\ \mathtt{C}_2$	by 22.3 and IH_{GEN}
$5: \Gamma \vdash \text{value-equal}(u_1, u_2) : \mathtt{Bool}^1$	by 22.2, 22.4, and def. of value-equal
$\therefore \mathcal{D}_{[\text{Eq}]}$ maintains IH_{GEN}	

Proof 23. $\mathcal{D}_{[\text{Seq}]}$	
$1: E \vdash e_1 \Downarrow v_1$	by inversion and $\mathcal{E}_{[\text{Seq}]}$
$2: \Gamma \vdash v_1 : \tau_1,\ \mathtt{C}_1$	by IH_{GEN}
$3: E \vdash e_2 \Downarrow v_2$	by inversion and $\mathcal{E}_{[\text{Seq}]}$
$4: \Gamma \vdash v_2 : \tau_2,\ \mathtt{C}_2$	by IH_{GEN}
$5: v_1 = u_1, \ldots, u_{\mathtt{m}_1}$	by $\mathcal{D}_{[\text{Seq}]}$
$6: v_2 = u_1, \ldots, u_{\mathtt{m}_2}$	by $\mathcal{D}_{[\text{Seq}]}$
$7: v_1, v_2 = u_1, \ldots, u_{\mathtt{m}_1+\mathtt{m}_2}$	by 23.5 and 23.6

Proof 23. $\mathcal{D}_{[\text{SEQ}]}$	
$8: \Gamma \vdash v_1, v_2 : (\tau_1, \tau_2)^{\mathtt{n}}, \{\mathtt{n} = \mathtt{m}_1 + \mathtt{m}_2\} \cup \mathtt{C}_1 \cup \mathtt{C}_2$	by 23.1, 23.3, and 23.7
$\therefore \mathcal{D}_{[\text{SEQ}]}$ maintains IH_{GEN}	

Proof 24. $\mathcal{D}_{[\text{COND}]}$	
$1: E \vdash e_1 \Downarrow v_1$	by inversion and $\mathcal{E}_{[\text{COND1}]}$
$2: \Gamma \vdash v_1 : \tau_1,\ \mathtt{C}_1$	by 24.1 and IH_{GEN}
3 : label $\mathtt{b}$ is unique	by pred2
4 : sizes $\mathtt{b}$, $\mathtt{notb}$ unique	by 24.3 and pred2
$5: \tau_1,\ \mathtt{C}_1 <: <\tau_1, \tau_2>^{\mathtt{r}}, \{\mathtt{b} \times \mathtt{m}_1 + \mathtt{notb} \times \mathtt{m}_2\} \cup \mathtt{C}_b \cup \mathtt{C}_1 \cup \mathtt{C}_2$	by 24.2, 24.4, and $\mathcal{S}_{\mathtt{PR1}}$
$6: \tau_2,\ \mathtt{C}_2 <: <\tau_1, \tau_2>^{\mathtt{r}}, \{\mathtt{b} \times \mathtt{m}_1 + \mathtt{notb} \times \mathtt{m}_2\} \cup \mathtt{C}_b \cup \mathtt{C}_1 \cup \mathtt{C}_2$	by proof of 24.5
$7: \Gamma \vdash \mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \tau,\ \mathtt{C} \wedge E \vdash \mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \Downarrow v \Rightarrow \Gamma \vdash v : \tau,\ \mathtt{C}$	by 24.5 and 24.6
$\therefore \mathcal{D}_{[\text{COND}]}$ maintains IH_{GEN}	

Proof 25. $\mathcal{D}_{[\text{LET}]}$	
$1: E \vdash e_1 \Downarrow v_1$	by inversion and $\mathcal{E}_{[\text{LET}]}$
$2: \Gamma \vdash v_1 : \tau_1,\ \mathtt{C}_1$	by 25.1 and IH_{GEN}
$3: \Gamma \uplus \{x : \tau_1\} \vdash e_2 : \tau_2$	by $\mathcal{D}_{[\text{LET}]}$
$4: \Gamma \vdash e_2[v_1/x] : \tau_2,\ \mathtt{C}_1 \cup \mathtt{C}_2$	by 25.2, 25.3, and sub. lemma
$\therefore \mathcal{D}_{[\text{LET}]}$ maintains IH_{GEN}	

Proof 26. $\mathcal{D}_{[\text{CASE}]}$	
1 : let $((\tau_1,\ \mathtt{C}_1\ \text{if}\ u'^{1} \neq \emptyset) \mid (\tau_2,\ \mathtt{C}_2\ \text{if}\ \tau' \neq \emptyset))^{\mathtt{r}}, \mathtt{C} = \tau'',\ \mathtt{C}''$	result type from $\mathcal{D}_{[\text{CASE}]}$
$2: E \vdash e_0 \Downarrow v'$	by inversion, $\mathcal{E}_{[\text{CASE1}]}$, and $\mathcal{E}_{[\text{CASE2}]}$
$3: \Gamma \vdash v' : u^{1}, \mathtt{C}$	by IH_{GEN}
$4: v' \in \text{Dom}(p) \Leftrightarrow \Gamma \vdash v' : u'^{1}, \mathtt{C}$	by case lemma, 26.2, and 26.3
$5: \Gamma \uplus \{x : u'^{1}\} \vdash e_1 : \tau_1,\ \mathtt{C} \cup \mathtt{C}_1$	by $\mathcal{D}_{[\text{CASE}]}$
$6: \Gamma \vdash e_1[v'/x] : \tau_1,\ \mathtt{C} \cup \mathtt{C}_1$	by 26.4, 26.5, and sub. lemma
$7: \Gamma \vdash v_1 : \tau_1,\ \mathtt{C} \cup \mathtt{C}_1$	by $\mathcal{E}_{[\text{CASE1}]}$ and IH_{GEN}
$8: \tau_1,\ \mathtt{C} \cup \mathtt{C}_1 <: \tau'',\ \mathtt{C}''$	by $\mathcal{S}_{\text{CHC1}}$ and $\mathcal{S}_{\text{CON}}$
$9: v' \notin \text{Dom}(p) \Leftrightarrow \Gamma \vdash v' : \tau', \mathtt{C}$	by case lemma, 26.2, and 26.3
$10: \Gamma \uplus \{x : \tau'\} \vdash e_2 : \tau_2,\ \mathtt{C} \cup \mathtt{C}_2$	by $\mathcal{D}_{[\text{CASE}]}$
$11: \Gamma \vdash e_2[v'/x] : \tau_2,\ \mathtt{C} \cup \mathtt{C}_2$	by 26.9, 26.10, and sub. lemma
$12: \Gamma \vdash v_2 : \tau_2,\ \mathtt{C} \cup \mathtt{C}_2$	by $\mathcal{E}_{[\text{CASE2}]}$ and IH_{GEN}
$13: \tau_2,\ \mathtt{C} \cup \mathtt{C}_2 <: \tau'',\ \mathtt{C}''$	by $\mathcal{S}_{\text{CHC2}}$ and $\mathcal{S}_{\text{CON}}$
$\therefore \mathcal{D}_{[\text{CASE}]}$ maintains IH_{GEN}	

C.1 Substitution Lemma

The lemma: If $\Gamma \vdash v : \tau,\ \mathtt{C}$ and $\Gamma \uplus \{x : v\} \vdash e : \tau',\ \mathtt{C}'$ then $\Gamma \vdash e\,[v/x] : \tau',\ \mathtt{C} \cup \mathtt{C}'$. This is also the inductive hypothesis.

Proof 27. $e = \mathtt{()}$	
$1: \Gamma \vdash \mathtt{()} : \mathtt{()}^{0}$	by [NIL]
$2: \mathtt{()}[v/x] = \mathtt{()}$	because $\mathtt{()}$ has no free variables
$3: \Gamma \vdash \mathtt{()}[v/x] = \mathtt{()}^{0}$	by substitution in 27.1 with 27.2
$\therefore e = \mathtt{()}$ maintains IH_{SUB}	

Proof 28. $e = \mathtt{c}_{\text{int}}$	
$1 : \Gamma \vdash \mathtt{c}_{\text{int}} : \mathtt{Integer}^1$	by [NIL]
$2 : \mathtt{c}_{\text{int}}[v/x] = \mathtt{c}_{\text{int}}$	because $\mathtt{c}_{\text{int}}$ has no free variables
$3 : \Gamma \vdash \mathtt{c}_{\text{int}}[v/x] = \mathtt{Integer}^1$	by substitution in 28.1 with 28.2
$\therefore e = \mathtt{c}_{\text{int}}$ maintains IH_{SUB}	

Proof 29. $e = x'$	
1 : Suppose $x \neq x'$	
$2 : \Gamma \uplus \{x : \tau\} \vdash x' : \tau', \mathtt{C}'$	assumption from proof
$3 : \Gamma \vdash x' : \tau', \mathtt{C}'$	by 29.2 and because $x \neq x'$
$4 : x'[v/x] = x'$	because x is not a free variable in x'
$5 : \Gamma \vdash x'[v/x] : x', \mathtt{C}'$	by substitution in 29.3 with 29.4
6 : Suppose $x = x'$	
$7 : \Gamma \uplus \{x : \tau\} \vdash x' : \tau', \mathtt{C}'$	assumption from proof
$8 : \Gamma \uplus \{x : \tau\}(x') = \tau', \mathtt{C}$	by def. of Γ
$9 : \Gamma \uplus \{x : \tau\} \vdash x' : \tau, \mathtt{C}$	by 29.8 and [VAR]
$10 : \tau = \tau', \mathtt{C} \cup \mathtt{C}'$	by transitivity on 29.7 and 29.9
$11 : \Gamma \vdash v : \tau, \mathtt{C}$	assumption from proof
$12 : x'[v/x] = v$	by semantics of [VAR]
$13 : \Gamma \vdash x'[v/x] : \tau', \mathtt{C} \cup \mathtt{C}'$	by substitution in 29.7, 29.9, and 29.12
$\therefore e = x'$ maintains IH_{SUB}	

Proof 30. $e = \mathtt{children}(e')$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $1 : \Gamma \uplus \{x : \tau\} \vdash e' : a[\tau']^1, \mathtt{C}'$ $\Rightarrow \Gamma \vdash e'[v/x] : a[\tau']^1, \mathtt{C} \cup \mathtt{C}'$	by inductive hypothesis and hypothesis of [CHILD]
$2 : \Gamma \vdash \mathtt{children}(e'[v/x]) : \tau', \mathtt{C} \cup \mathtt{C}'$	by 30.1 and [CHILD]
$3 : \Gamma \vdash \mathtt{children}(e')[v/x] : \tau', \mathtt{C} \cup \mathtt{C}'$	by 30.2 and distributivity
$\therefore e = \mathtt{children}(e')$ maintains IH_{SUB}	

Proof 31. $e = a[e']$	
1 : Let $\tau' = a[\tau''], \mathtt{C} \cup \mathtt{C}'$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e' : \tau'', \mathtt{C}'$ $\Rightarrow \Gamma \vdash e'[v/x] : \tau'', \mathtt{C} \cup \mathtt{C}'$	by inductive hypothesis and hypothesis of [ELT]
$3 : \Gamma \vdash a[e'[v/x]] : \tau', \mathtt{C} \cup \mathtt{C}'$	by 31.2 and [ELT]
$4 : \Gamma \vdash [e'][v/x] : \tau', \mathtt{C} \cup \mathtt{C}'$	by 31.3 and distributivity
$\therefore e = a[e']$ maintains IH_{SUB}	

Proof 32. $e = e_1 + e_2$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $1 : \Gamma \uplus \{x : \tau\} \vdash e_1 : \mathtt{Integer}^1, \mathtt{C}'$ $\Rightarrow \Gamma \vdash e_1[v/x] : \mathtt{Integer}^1, \mathtt{C} \cup \mathtt{C}'$	by inductive hypothesis and hypothesis of [PLUS]
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e_2 : \mathtt{Integer}^1, \mathtt{C}'$ $\Rightarrow \Gamma \vdash e_2[v/x] : \mathtt{Integer}^1, \mathtt{C} \cup \mathtt{C}'$	by inductive hypothesis and hypothesis of [PLUS]
$3 : \Gamma \vdash (e_1[v/x]) + (e_2[v/x]) : \mathtt{Integer}^1, \mathtt{C} \cup \mathtt{C}'$	by 32.1, 32.2, and [PLUS]
$4 : \Gamma \vdash (e_1 + e_2)[v/x] : \mathtt{Integer}^1, \mathtt{C} \cup \mathtt{C}'$	by 32.3 and distributivity
$\therefore e = e_1 + e_2$ maintains IH_{SUB}	

Proof 33. $e = e_1 + e_2$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $1 : \Gamma \uplus \{x : \tau\} \vdash e_1 : u^1, \mathtt{C}'$ $\Rightarrow \Gamma \vdash e_1[v/x] : u^1, \mathtt{C} \cup \mathtt{C}'$	by inductive hypothesis and hypothesis of [EQ]

Proof 33. $e = e_1 + e_2$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e_2 : u^1, \mathtt{C'}$ $\Rightarrow \Gamma \vdash e_2[v/x] : u^1, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [EQ]
$3 : \Gamma \vdash (e_1[v/x]) = (e_2[v/x]) : \mathtt{Boolean}^1, \mathtt{C} \cup \mathtt{C'}$	by 33.1, 33.2, and [EQ]
$4 : \Gamma \vdash (e_1 = e_2)[v/x] : \mathtt{Boolean}^1, \mathtt{C} \cup \mathtt{C'}$	by 33.3 and distributivity
$\therefore e = e_1 + e_2$ maintains $\mathrm{IH_{SUB}}$	

Proof 34. $e = e_1, e_2$	
$1 :$ Let $\tau' = (\tau_1, \tau_2)^{\mathrm{n}}, \mathtt{C} \cup \mathtt{C'}$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e_1 : \tau_1, \mathtt{C'}$ $\Rightarrow \Gamma \vdash e_1[v/x] : \tau_1, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [SEQ]
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $3 : \Gamma \uplus \{x : \tau\} \vdash e_2 : \tau_2, \mathtt{C'}$ $\Rightarrow \Gamma \vdash e_2[v/x] : \tau_2, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [SEQ]
$4 : \Gamma \vdash (e_1[v/x]), (e_2[v/x]) : \tau', \mathtt{C} \cup \mathtt{C'}$	by 34.2, 34.3, and [SEQ]
$5 : \Gamma \vdash (e_1, e_2)[v/x] : \tau', \mathtt{C} \cup \mathtt{C'}$	by 34.4 and distributivity
$\therefore e = e_1, e_2$ maintains $\mathrm{IH_{SUB}}$	

Proof 35. $e = \mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2$	
$1 :$ Let $\tau' = \mathtt{<}\tau_1, \tau_2\mathtt{>}^{\mathrm{n}}, \mathtt{C} \cup \mathtt{C'}$ and $\mathtt{if}.label = \mathtt{b}$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e_b : \mathtt{Boolean}^1, \mathtt{C'}$ $\Rightarrow \Gamma \vdash e_1[v/x] : \mathtt{Boolean}^1, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [COND]
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $3 : \Gamma \uplus \{x : \tau\} \vdash e_1 : \tau_1, \mathtt{C'}$ $\Rightarrow \Gamma \vdash e_1[v/x] : \tau_1, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [COND]
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $4 : \Gamma \uplus \{x : \tau\} \vdash e_2 : \tau_2, \mathtt{C'}$ $\Rightarrow \Gamma \vdash e_2[v/x] : \tau_2, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [COND]
$5 : \Gamma \vdash \mathtt{if}\ (e_b[v/x])\ \mathtt{then}\ (e_1[v/x])$ $\mathtt{else}\ (e_2[v/x]) : \tau', \mathtt{C} \cup \mathtt{C'}$	by 35.2, 35.3, 35.4, and [SEQ]
$6 : \Gamma \vdash (\mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2)[v/x] : \tau',$ $\mathtt{C} \cup \mathtt{C'} \cup \{\mathtt{n} = (\mathtt{b} \times \mathtt{n}_1 + \mathtt{notb} \times \mathtt{n}_2)\}$	by 35.5 and distributivity and because $\mathtt{b}$ and $\mathtt{notb}$ are unbound and $\mathtt{n}_1$ and $\mathtt{n}_2$ are the branches' dimensions
$\therefore e = \mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2$ maintains $\mathrm{IH_{SUB}}$	

Proof 36. $e = \mathtt{case}\ e_0\ \mathtt{of}\ x'\mathtt{:}p\ \mathtt{=>}\ e_1\ \mid\ x_2\ \mathtt{=>}\ e_2$	
$1 :$ Let $\tau' = (\tau_1, \tau_2)^{\mathrm{n}}, \mathtt{C} \cup \mathtt{C'}$	
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e_0 : u^1, \mathtt{C'}$ $\Rightarrow \Gamma \vdash e_0[v/x] : u^1, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [CASE]
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $3 : \Gamma \uplus \{x' : u'^1\} \uplus \{x : \tau\} \vdash e_1 : \tau_1, \mathtt{C'}$ $\Rightarrow \Gamma \uplus \{x' : u'^1\} \vdash e_1[v/x] : \tau_1, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [CASE]
$\Gamma \vdash v : \tau, \mathtt{C} \wedge$ $4 : \Gamma \uplus \{x2 : \tau'\} \uplus \{x : \tau\} \vdash e_2 : \tau_2, \mathtt{C'}$ $\Rightarrow \Gamma \uplus \{x2 : \tau'\} \vdash e_2[v/x] : \tau_2, \mathtt{C} \cup \mathtt{C'}$	by inductive hypothesis and hypothesis of [CASE]
$5 : \Gamma \vdash \mathtt{case}\ (e[v/x])\ \mathtt{of}\ x'\mathtt{:}p\ \mathtt{=>}\ (e_1[v/x])$ $\mid\ x2\ \mathtt{=>}\ (e_2[v/x]) : \tau', \mathtt{C} \cup \mathtt{C'}$	by 36.2, 36.3, 36.4, and [CASE]
$6 : \Gamma \vdash (\mathtt{case}\ e_0\ \mathtt{of}\ x'\mathtt{:}p\ \mathtt{=>}\ e_1$ $\mid\ x2\ \mathtt{=>}\ e_2)[v/x] : \tau', \mathtt{C} \cup \mathtt{C'}$	by 36.5 and distributivity
$\therefore e = \mathtt{case}\ e_0\ \mathtt{of}\ x'\mathtt{:}p\ \mathtt{=>}\ e_1\ \mid\ x_2\ \mathtt{=>}\ e_2$ maintains $\mathrm{IH_{SUB}}$	

Proof 37. $e =$ `let` x' `=` e_1 `in` e_2	
$\Gamma \vdash v : \tau, \texttt{C} \wedge$ $1 : \Gamma \uplus \{x : \tau\} \vdash e_1 : \tau_1, \texttt{C}'$ $\Rightarrow \Gamma \vdash e_1[v/x] : \tau_1, \texttt{C} \cup \texttt{C}'$	by inductive hypothesis and hypothesis of [LET]
$\Gamma \vdash v : \tau, \texttt{C} \wedge$ $2 : \Gamma \uplus \{x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau', \texttt{C}'$ $\Rightarrow \Gamma \uplus \{x' : \tau_1\} \vdash e_2[v/x] : \tau', \texttt{C} \cup \texttt{C}'$	by inductive hypothesis and hypothesis of [LET]
$3 : \Gamma \vdash$ `let` x' `=` $(e_1[v/x])$ `in` $(e_2[v/x]) : \tau', \texttt{C} \cup \texttt{C}'$	by 37.1, 37.2, and [LET]
$4 : \Gamma \vdash ($`let` x' `=` e_1 `in` $e_2)[v/x] : \tau', \texttt{C} \cup \texttt{C}'$	by 37.3 and distributivity
$\therefore e =$ `let` x' `=` e_1 `in` e_2 maintains IH_{SUB}	

Proof 38. $e =$ `for` x' `in` e_1 `do` e_2	
1 : Suppose `for`.$label = \emptyset$	
$\Gamma \vdash v : \tau, \texttt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e_1 : \tau_1, \texttt{C}'$ $\Rightarrow \Gamma \vdash e_1[v/x] : \tau_1, \texttt{C} \cup \texttt{C}'$	by inductive hypothesis and hypothesis of [FOR]
$\Gamma \vdash v : \tau, \texttt{C} \wedge$ $3 : \Gamma \uplus \{\texttt{for}\, x : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_2, \texttt{C}'$ $\Rightarrow \Gamma \vdash e_2[v/x] : \tau_2, \texttt{C} \cup \texttt{C}'$	by inductive hypothesis, proof of $\text{IH}_{\text{SUB}\Pi}$, and hypothesis of [FOR]
$4 : \Gamma \vdash$ `for` x' `in` $(e_1[v/x])$ `do` $(e_2[v/x]) : \tau', \texttt{C} \cup \texttt{C}'$	by 38.2, 38.3, and [FOR]
$5 : \Gamma \vdash ($`for` x' `in` e_1 `do` $e_2)[v/x] : \tau', \texttt{C} \cup \texttt{C}'$	by 38.4 and distributivity
$\therefore e =$ `for` x' `in` e_1 `do` e_2 maintains IH_{SUB}	

Proof 39. $e =$ `for` x' `in` e_1 `do` e_2	
1 : Suppose `for`.$label = start$ and $\exists \pi.\, \pi \in \texttt{C}'$ and $\tau_2 = (z^{\texttt{m}})^{*\texttt{n}}$	
$\Gamma \vdash v : \tau, \texttt{C} \wedge$ $2 : \Gamma \uplus \{x : \tau\} \vdash e_1 : \tau_1, \texttt{C}'$ $\Rightarrow \Gamma \vdash e_1[v/x] : \tau_1, \texttt{C} \cup \texttt{C}'$	by inductive hypothesis and hypothesis of [FORΠ]
$\Gamma \vdash v : \tau, \texttt{C} \wedge$ $3 : \Gamma \uplus \{\texttt{for}\, x : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_2, \texttt{C}'$ $\Rightarrow \Gamma \vdash e_2[v/x] : \tau_2, \texttt{C} \cup \texttt{C}'$	by inductive hypothesis, proof of $\text{IH}_{\text{SUB}\Pi}$, and hypothesis of [FORΠ]
$\Gamma \vdash$ `for` x' `in` $(e_1[v/x])$ $4 :$ `do` $(e_2[v/x]) : \tau', ((\texttt{C} \cup \texttt{C}') \setminus \{\pi\}) \cup$ $\{\texttt{n} = \texttt{m} \times \Gamma_\pi(\pi)\}$	By 39.2, 39.3, and [FORΠ]. Pred1 requires that this expression be at the "top" of path π and other iterations, and none of the rules the expressions may pass through ([FOR], [CASE], [FORΠ*], [FORU]) add constraints to n
$5 : \Gamma \vdash ($`for` x' `in` e_1 `do` $e_2)[v/x] : \tau'$, $((\texttt{C} \cup \texttt{C}') \setminus \{\pi\}) \cup \{\texttt{n} = \texttt{m} \times \Gamma_\pi(\pi)\}$	by 39.4 and distributivity
$\therefore e =$ `for` x' `in` e_1 `do` e_2 maintains IH_{SUB}	

$\text{IH}_{\text{SUBFOR}} = (\Gamma \vdash v : \tau, \texttt{C} \wedge \Gamma \uplus \{\texttt{for}\, x : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_2, \texttt{C}' \Rightarrow \Gamma \uplus \{\texttt{for}\, x : \tau_1\} \vdash e_2[v/x] : \tau_2, \texttt{C} \cup \texttt{C}')$
This proof is by induction on the type structure of τ_1.

Proof 40. $\tau_1 = u^1$	
1 : If $x.label = \emptyset$:	
$2 : \Gamma \uplus \{x' : u^1\} \vdash v : \tau, \texttt{C} \wedge \Gamma \uplus \{x' : u^1\} \uplus \{x : \tau\}\Gamma \vdash e_2 : \tau_2, \texttt{C}'$ $\Rightarrow \Gamma \uplus \{x' : u^1\} \vdash e_2[v/x] : \tau_2, \texttt{C} \cup \texttt{C}'$	by the IH_{SUB} on hypothesis of [FORU]
$\Gamma \uplus \{\texttt{for}\, x' : u^1\} \vdash v : \tau, \texttt{C}$ $3 : \wedge \Gamma \uplus \{\texttt{for}\, x' : u^1\} \uplus \{x : \tau\} \vdash e_2 : \tau_2, \texttt{C}'$ $\Rightarrow \Gamma \uplus \{\texttt{for}\, x' : u^1\} \vdash e_2 : \tau_2, \texttt{C} \cup \texttt{C}'$	by 40.2 and [FORU]
4 : If $x.label = \pi$:	
$5 : \Gamma \uplus \{x' : u^1\} \vdash v : \tau, \texttt{C} \wedge \Gamma \uplus \{x' : u^1\} \uplus \{x : \tau\}\Gamma \vdash e_2 : \tau_2, \texttt{C}'$ $\Rightarrow \Gamma \uplus \{x' : u^1\} \vdash e_2[v/x] : \tau_2, \texttt{C} \cup \texttt{C}'$	by the IH_{SUB} on hypothesis of [FORΠU]
$\Gamma \uplus \{\texttt{for}\, x' : u^1\} \vdash v : \tau, \texttt{C}$ $6 : \wedge \Gamma \uplus \{\texttt{for}\, x' : u^1\} \uplus \{x : \tau\} \vdash e_2 : \tau_2^{*\texttt{n}}, \texttt{C}' \cup \{\pi\}$ $\Rightarrow \Gamma \uplus \{\texttt{for}\, x' : u^1\} \vdash e_2 : \tau_2^{*\texttt{n}}, \texttt{C} \cup \texttt{C}' \cup \{\pi\}$	by 40.5 and [FORΠU]
$\therefore \tau_1 = u^1$ maintains $\text{IH}_{\text{SUBFOR}}$	

Proof 41. $\tau_1 = \texttt{()}^0$	
$1: \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \texttt{()}^0\} \vdash v : \tau, \texttt{C} \wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \texttt{()}^0\} \uplus \{x : \tau\} \vdash e_2 : \texttt{()}^0$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \texttt{()}^0\} \vdash e_2[v/x] : \tau_2, \texttt{C}$	by [FORN]
$\therefore \tau_1 = \texttt{()}^0$ maintains $\text{IH}_{\text{SUBFOR}}$	

Proof 42. $\tau_1 = \emptyset$	
$1: \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \emptyset\} \vdash v : \tau, \texttt{C} \wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \emptyset\} \uplus \{x : \tau\} \vdash e_2 : \emptyset$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \emptyset\} \vdash e_2[v/x] : \emptyset$	by [FORN]
$\therefore \tau_1 = \emptyset$ maintains $\text{IH}_{\text{SUBFOR}}$	

Proof 43. $\tau_1 = (\tau_{1a}, \tau_{1b})^{\text{n}}$	
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_{1a}\} \vdash v : \tau, \texttt{C}$ $1: \wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_{1a}\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}, \texttt{C}'_1$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_{1a}\} \vdash e_2 : \tau_{2a}, \texttt{C} \cup \texttt{C}'_1$	by $\text{IH}_{\text{SUBFOR}}$
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_{1b}\} \vdash v : \tau, \texttt{C}$ $2: \wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_{1b}\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b}, \texttt{C}'_2$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_{1b}\} \vdash e_2 : \tau_{2b}, \texttt{C} \cup \texttt{C}'_2$	by $\text{IH}_{\text{SUBFOR}}$
3 : If $\neg\exists\pi.\, \pi \in \texttt{C}'_{1,2}$:	
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash v : \tau, \texttt{C}$ $\wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : (\tau_{2a}, \tau_{2b})^{\text{n}_2},$ $4: \quad \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2 \cup \{\text{n}_2 = \text{m}_1 + \text{m}_2\}$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash e_2[v/x] : (\tau_{2a}, \tau_{2b})^{\text{n}_2},$ $\texttt{C} \cup \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2 \cup \{\text{n}_2 = \text{m}_1 + \text{m}_2\}$	by 43.1, 43.2, and [FORS] where m_1 and m_2 are the dimension annotations of τ_{2a} and τ_{2b} respectively
5 : If $\exists\pi.\, \pi \in \texttt{C}'_{1,2}$:	
$6: (\tau_{2a} \neq \texttt{()}^0 \neq \tau_{2b}) \wedge (\tau_{2a} \neq \emptyset \neq \tau_{2b})$ $\Rightarrow \tau_{2a} = \tau_{2b}, \texttt{C}'_1 \cup \texttt{C}'_2$	by the hypothesis of [FORΠ1]
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash v : \tau, \texttt{C}$ $\wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a},$ $7: \quad \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a},$ $\texttt{C} \cup \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$	by 43.1, 43.2, 43.6, and [FORΠ1]
8 : If $(\tau_{2b} = \texttt{()}^0) \wedge (\emptyset \neq \tau_{2a} \neq \texttt{()}^0)$:	
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash v : \tau, \texttt{C}$ $\wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a},$ $9: \quad \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a},$ $\texttt{C} \cup \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$	by 43.1, 43.2, 43.8, and [FORΠ2]
10 : If $(\tau_{2a} = \texttt{()}^0) \wedge (\emptyset \neq \tau_{2b} \neq \texttt{()}^0)$:	
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash v : \tau, \texttt{C}$ $\wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b},$ $11: \quad \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash e_2[v/x] : \tau_{2b},$ $\texttt{C} \cup \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$	by 43.1, 43.2, 43.10, and [FORΠ3]
12 : If $(\tau_{2b} = \emptyset) \wedge (\tau_{2a} \neq \emptyset)$:	
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash v : \tau, \texttt{C}$ $\wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a},$ $13: \quad \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a},$ $\texttt{C} \cup \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$	by 43.1, 43.2, 43.12, and [FORΠ4]
14 : If $(\tau_{2a} = \emptyset) \wedge (\tau_{2b} \neq \emptyset)$:	
$\mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash v : \tau, \texttt{C}$ $\wedge \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b},$ $15: \quad \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$ $\Rightarrow \mathit{\Gamma} \uplus \{\texttt{for}\, x' : \tau_1\} \vdash e_2[v/x] : \tau_{2b},$ $\texttt{C} \cup \texttt{C}' \cup \texttt{C}'_1 \cup \texttt{C}'_2$	by 43.1, 43.2, 43.14, and [FORΠ5]
$\therefore \tau_1 = (\tau_{1a}, \tau_{1b})^{\text{n}}$ maintains $\text{IH}_{\text{SUBFOR}}$	

Proof 44. $\tau_1 = (\tau_{1a} \vert \tau_{1b})^{\text{n}}$

Proof 44. $\tau_1 = (\tau_{1a} \vert \tau_{1b})^{\mathtt{n}}$		
1 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_{1a}\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1a}\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\, ,\ \mathtt{C}'_1$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1a}\} \vdash e_2 : \tau_{2a}\, ,\ \mathtt{C} \cup \mathtt{C}'_1$	by $\mathrm{IH}_{\mathrm{SUBFOR}}$
2 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_{1b}\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1b}\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b}\, ,\ \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1b}\} \vdash e_2 : \tau_{2b}\, ,\ \mathtt{C} \cup \mathtt{C}'_2$	by $\mathrm{IH}_{\mathrm{SUBFOR}}$
3 :	If $\neg\exists\pi.\ \pi \in \mathtt{C}'_{1,2}$:	
4 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : (\tau_{2a}, \tau_{2b})^{\mathtt{n}_2}\, ,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash e_2[v/x] : (\tau_{2a}, \tau_{2b})^{\mathtt{n}_2}\, ,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 44.1, 44.2, and [ForC]
5 :	If $\exists\pi.\ \pi \in \mathtt{C}'_{1,2}$:	
6 :	$(\tau_{2a} \neq (\,)^0 \neq \tau_{2b})\ \wedge\ (\tau_{2a} \neq \emptyset \neq \tau_{2b})$ $\Rightarrow\ \tau_{2a} = \tau_{2b}\, ,\ \mathtt{C}'_1 \cup \mathtt{C}'_2$	by the hypothesis of [ForΠ1]
7 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\, ,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}\, ,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 44.1, 44.2, 44.6, and [ForΠ1]
8 :	If $(\tau_{2b} = (\,)^0)\ \wedge\ (\emptyset \neq \tau_{2a} \neq (\,)^0)$:	
9 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\, ,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}\, ,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 44.1, 44.2, 44.8, and [ForΠ2]
10 :	If $(\tau_{2a} = (\,)^0)\ \wedge\ (\emptyset \neq \tau_{2b} \neq (\,)^0)$:	
11 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b}\, ,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2b}\, ,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 44.1, 44.2, 44.10, and [ForΠ3]
12 :	If $(\tau_{2b} = \emptyset)\ \wedge\ (\tau_{2a} \neq \emptyset)$:	
13 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\, ,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}\, ,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 44.1, 44.2, 44.12, and [ForΠ4]
14 :	If $(\tau_{2a} = \emptyset)\ \wedge\ (\tau_{2b} \neq \emptyset)$:	
15 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b}\, ,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2b}\, ,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 44.1, 44.2, 44.14, and [ForΠ5]
$\therefore\ \tau_1 = (\tau_{1a} \vert \tau_{1b})^{\mathtt{n}}$ maintains $\mathrm{IH}_{\mathrm{SUBFOR}}$		

Proof 45. $\tau_1 = \mathtt{<}\tau_{1a}, \tau_{1b}\mathtt{>}^{\mathtt{n}}$		
1 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_{1a}\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1a}\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\, ,\ \mathtt{C}'_1$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1a}\} \vdash e_2 : \tau_{2a}\, ,\ \mathtt{C} \cup \mathtt{C}'_1$	by $\mathrm{IH}_{\mathrm{SUBFOR}}$
2 :	$\Gamma \uplus \{\mathtt{for}\ x' : \tau_{1b}\} \vdash v : \tau\, ,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1b}\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b}\, ,\ \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\mathtt{for}\ x' : \tau_{1b}\} \vdash e_2 : \tau_{2b}\, ,\ \mathtt{C} \cup \mathtt{C}'_2$	by $\mathrm{IH}_{\mathrm{SUBFOR}}$
3 :	If $\neg\exists\pi.\ \pi \in \mathtt{C}'_{1,2}$:	
4 :	$\mathtt{n} = \mathtt{m}'_1 \times \mathtt{b} + \mathtt{m}'_2 \times \mathtt{notb}$	by the hypothesis of [ForP] where $\mathtt{m}'_1$ and $\mathtt{m}'_2$ are the dimension annotations of τ_{1a} and τ_{1b} respectively

Proof 45. $\tau_1 = \texttt{<}\tau_{1a}, \tau_{1b}\texttt{>}^{\mathrm{n}}$		
5 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : (\tau_{2a}, \tau_{2b})^{\mathtt{n}_2}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2 \cup \{\mathtt{n}_2 = \mathtt{m}_1 \times \mathtt{b} + \mathtt{m}_2 \times \mathtt{notb}\}$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : (\tau_{2a}, \tau_{2b})^{\mathtt{n}_2}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2 \cup \{\mathtt{n}_2 = \mathtt{m}_1 \times \mathtt{b} + \mathtt{m}_2 \times \mathtt{notb}\}$	by 45.1, 45.2, and [ForP] where $\mathtt{m}_1$ and $\mathtt{m}_2$ are the dimension annotations of τ_{2a} and τ_{2b} respectively
6 :	If $\exists \pi .\ \pi \in \mathtt{C}'_{1,2}$:	
7 :	$(\tau_{2a} \neq (\,)^{0} \neq \tau_{2b})\ \wedge\ (\tau_{2a} \neq \emptyset \neq \tau_{2b})$ $\Rightarrow\ \tau_{2a} = \tau_{2b}\,,\ \mathtt{C}'_1 \cup \mathtt{C}'_2$	by the hypothesis of [ForΠ1]
8 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 45.1, 45.2, 45.7, and [ForΠ1]
9 :	If $(\tau_{2b} = (\,)^{0})\ \wedge\ (\emptyset \neq \tau_{2a} \neq (\,)^{0})$:	
10 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 45.1, 45.2, 45.9, and [ForΠ2]
11 :	If $(\tau_{2a} = (\,)^{0})\ \wedge\ (\emptyset \neq \tau_{2b} \neq (\,)^{0})$:	
12 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2b}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 45.1, 45.2, 45.11, and [ForΠ3]
13 :	If $(\tau_{2b} = \emptyset)\ \wedge\ (\tau_{2a} \neq \emptyset)$:	
14 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 45.1, 45.2, 45.13, and [ForΠ4]
15 :	If $(\tau_{2a} = \emptyset)\ \wedge\ (\tau_{2b} \neq \emptyset)$:	
16 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2b}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2b}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \mathtt{C}'_2$	by 45.1, 45.2, 45.15, and [ForΠ5]
$\therefore \tau_1 = \texttt{<}\tau_{1a}, \tau_{1b}\texttt{>}^{\mathrm{n}}$ maintains $\mathrm{IH}_{\mathrm{SUBFOR}}$		

Proof 46. $\tau_1 = \tau_{1a}^{*\mathrm{n}}$		
1 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_{1a}\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_{1a}\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\,,\ \mathtt{C}'_1$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_{1a}\} \vdash e_2 : \tau_{2a}\,,\ \mathtt{C} \cup \mathtt{C}'_1$	by $\mathrm{IH}_{\mathrm{SUBFOR}}$
2 :	If $\neg\exists \pi .\ \pi \in \mathtt{C}'_1$:	
3 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}^{*\mathtt{n}_2}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1 \cup \{\mathtt{n}_2 = \mathtt{n}/\mathtt{m} \times \mathtt{m}_1\}$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}^{*\mathtt{n}_2}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1 \cup \{\mathtt{n}_2 = \mathtt{n}/\mathtt{m} \times \mathtt{m}_2\}$	by 46.1 and [ForR] where $\mathtt{m}$ and $\mathtt{m}_2$ are the dimension annotations of τ_{1a} and τ_{2a} respectively
4 :	If $\exists \pi .\ \pi \in \mathtt{C}'_{1,2}$:	
5 :	$\Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash v : \tau\,,\ \mathtt{C}$ $\wedge\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \uplus \{x : \tau\} \vdash e_2 : \tau_{2a}\,,$ $\mathtt{C}' \cup \mathtt{C}'_1$ $\Rightarrow\ \Gamma \uplus \{\texttt{for}\ x' : \tau_1\} \vdash e_2[v/x] : \tau_{2a}\,,$ $\mathtt{C} \cup \mathtt{C}' \cup \mathtt{C}'_1$	by 46.1, 46.4, and [ForΠR]
$\therefore \tau_1 = \tau_{1a}^{*\mathrm{n}}$ maintains $\mathrm{IH}_{\mathrm{SUBFOR}}$		

C.2 Case Lemma

We prove the following hypothesis by structural induction for any unit type τ:
For $\Gamma \vdash v : \tau$ and $split^p(\tau) = {u'}^1 | \tau'$, $v \in \mathrm{Dom}(p) \Leftrightarrow \Gamma \vdash v : {u'}^1$ and $v \notin \mathrm{Dom}(p) \Leftrightarrow \Gamma \vdash v : \tau'$.

C.2.1 *case:* $p = a$

case: $\tau = s$: $\neg\exists v.(\Gamma \vdash v : s \wedge v \in \mathrm{Dom}(p))$
$(\Gamma \vdash v : s \Leftrightarrow v \in \mathrm{Dom}(s)) \wedge (\mathrm{Dom}(s) \cap \mathrm{Dom}(p) = \emptyset)$

case: $\tau = a[\tau]$: $(\Gamma \vdash v : a[\tau] \Leftrightarrow v \in \mathrm{Dom}(p))$
$\neg\exists v.(v \in \mathrm{Dom}(p) \wedge v \notin \mathrm{Dom}(a))$

case: $\tau = a'[\tau]$: $\neg\exists v.(v \in \mathrm{Dom}(p) \wedge v \notin \mathrm{Dom}(a'))$
$(\Gamma \vdash v : a'[\tau] \Leftrightarrow v \in \mathrm{Dom}(a')) \wedge (\mathrm{Dom}(a') \cap \mathrm{Dom}(p) = \emptyset)$

case: $\tau = \tilde{}[\tau]$: $a[\tau] <: \tilde{}[\tau] \Rightarrow \exists v.(\Gamma \vdash v : \tilde{}[\tau] \wedge \Gamma \vdash v : a[\tau])$
$\wedge (\Gamma \vdash v : a[\top] \Leftrightarrow v \in \mathrm{Dom}(p))$
$\Rightarrow \exists v.(\Gamma \vdash v : \tilde{}[\tau] \wedge v \in \mathrm{Dom}(p))$
$\neg(\tilde{}[\tau] <: a[\tau]) \Rightarrow \exists v.(\Gamma \vdash v : \tilde{}[\tau] \wedge \Gamma \vdash v : a'[\tau])$
$\wedge (\Gamma \vdash v : a'[\top] \Leftrightarrow v \notin \mathrm{Dom}(p))$
$\Rightarrow \exists v.(\Gamma \vdash v : \tilde{}[\tau] \wedge v \notin \mathrm{Dom}(p))$

case: $\tau = \tau_1 \mid \tau_2$: If $\Gamma \vdash v : \tau_1$, the inductive hypothesis on τ_1 proves the lemma for this case.
Otherwise, $\Gamma \vdash v : \tau_2$, and the inductive hypothesis on τ_2 proves the lemma for this case.

case: $\tau = <\tau_1, \tau_2>$: If $\Gamma \vdash v : \tau_1$, the inductive hypothesis on τ_1 proves the lemma for this case.
Otherwise, $\Gamma \vdash v : \tau_2$, and the inductive hypothesis on τ_2 proves the lemma for this case.

case: $\tau = \emptyset$: $\neg\exists v.(\Gamma \vdash v : \emptyset) \Rightarrow \neg\exists v.(\Gamma \vdash v : \emptyset \wedge v \in \mathrm{Dom}(p))$
$\neg\exists v.(\Gamma \vdash v : \emptyset) \Rightarrow \neg\exists v.(\Gamma \vdash v : \emptyset \wedge v \notin \mathrm{Dom}(p))$

C.2.2 *case:* $p = \tilde{}$

case: $\tau = s$: $\neg\exists v.(\Gamma \vdash v : s \wedge v \in \mathrm{Dom}(p))$
$(\Gamma \vdash v : s \Leftrightarrow v \in \mathrm{Dom}(s)) \wedge (\mathrm{Dom}(s) \cap \mathrm{Dom}(p) = \emptyset)$

case: $\tau = a[\tau]$: $((a[\tau] <: \tilde{}[\tau] \wedge \Gamma \vdash v : a[\tau]) \Leftrightarrow v \in \mathrm{Dom}(p))$
$\neg(\tilde{}[\tau] <: a[\tau]) \Rightarrow \neg\exists v.(v \in \mathrm{Dom}(p) \wedge v \notin \mathrm{Dom}(a))$

case: $\tau = \tilde{}[\tau]$: $(\Gamma \vdash v : \tilde{}[\tau] \Leftrightarrow v \in \mathrm{Dom}(p))$
$\neg\exists v.(v \in \mathrm{Dom}(p) \wedge v \notin \mathrm{Dom}(\tilde{}))$

case: $\tau = \tau_1 \mid \tau_2$: If $\Gamma \vdash v : \tau_1$, the inductive hypothesis on τ_1 proves the lemma for this case.
Otherwise, $\Gamma \vdash v : \tau_2$, and the inductive hypothesis on τ_2 proves the lemma for this case.

case: $\tau = <\tau_1, \tau_2>$: If $\Gamma \vdash v : \tau_1$, the inductive hypothesis on τ_1 proves the lemma for this case.
Otherwise, $\Gamma \vdash v : \tau_2$, and the inductive hypothesis on τ_2 proves the lemma for this case.

case: $\tau = \emptyset$: $\neg\exists v.(\Gamma \vdash v : \emptyset) \Rightarrow \neg\exists v.(\Gamma \vdash v : \emptyset \wedge v \in \mathrm{Dom}(p))$
$\neg\exists v.(\Gamma \vdash v : \emptyset) \Rightarrow \neg\exists v.(\Gamma \vdash v : \emptyset \wedge v \notin \mathrm{Dom}(p))$

C.2.3 *case:* $p = s$

case: $\tau = s'$: $((s' <: s) \Rightarrow (\mathrm{Dom}(s') \subseteq \mathrm{Dom}(s)) \Rightarrow (v \in \mathrm{Dom}(p)))$
$\wedge ((s <: s') \Rightarrow (\exists v.(\Gamma \vdash v : s' \wedge \Gamma \vdash v : s) \wedge (\Gamma \vdash v : s \Leftrightarrow v \in \mathrm{Dom}(p))$
$\Rightarrow \exists v.(\Gamma \vdash v : s' \wedge v \in \mathrm{Dom}(p))))$
$\wedge ((\neg(s <: s') \wedge \neg(s' <: s)) \Rightarrow (\mathrm{Dom}(s) \cap \mathrm{Dom}(s')) \Rightarrow (v \notin \mathrm{Dom}(p)))$
$((s' <: s) \Rightarrow \neg\exists v.(v \in \mathrm{Dom}(s') \wedge v \notin \mathrm{Dom}(s)))$
$\wedge ((s <: s') \Rightarrow (\exists v.(\Gamma \vdash v : s' \wedge \Gamma \vdash v : s'') \wedge (\Gamma \vdash v : s'' \Leftrightarrow v \in \mathrm{Dom}(s''))$
$\wedge (\mathrm{Dom}(s) \cap \mathrm{Dom}(s'') = \emptyset)) \Rightarrow \exists v.(\Gamma \vdash v : s' \wedge v \notin \mathrm{Dom}(p))))$
$\wedge ((\neg(s <: s') \wedge \neg(s' <: s)) \Rightarrow (\mathrm{Dom}(s) \cap \mathrm{Dom}(s'))$
$\Rightarrow (v \notin \mathrm{Dom}(p) \wedge \Gamma \vdash v : s'))$

case: $\tau = a[\tau]$: $\neg\exists v.(\Gamma \vdash v : a[\tau] \wedge v \in \mathrm{Dom}(p))$
$\Gamma \vdash v : a[\tau] \Leftrightarrow v \in \mathrm{Dom}(p$

case: $\tau = \tilde{}[\tau]$: $\neg\exists v.(\Gamma \vdash v : \tilde{}[\tau] \wedge v \in \mathrm{Dom}(p))$
$\Gamma \vdash v : \tilde{}[\tau] \Leftrightarrow v \in \mathrm{Dom}(p$

case: $\tau = \tau_1 \mid \tau_2$: If $\Gamma \vdash v : \tau_1$, the inductive hypothesis on τ_1 proves the lemma for this case.
Otherwise, $\Gamma \vdash v : \tau_2$, and the inductive hypothesis on τ_2 proves the lemma for this case.

case: $\tau = <\tau_1, \tau_2>$: If $\Gamma \vdash v : \tau_1$, the inductive hypothesis on τ_1 proves the lemma for this case.
Otherwise, $\Gamma \vdash v : \tau_2$, and the inductive hypothesis on τ_2 proves the lemma for this case.

case: $\tau = \emptyset$: $\neg\exists v.(\Gamma \vdash v : \emptyset) \Rightarrow \neg\exists v.(\Gamma \vdash v : \emptyset \wedge v \in \mathrm{Dom}(p))$
$\neg\exists v.(\Gamma \vdash v : \emptyset) \Rightarrow \neg\exists v.(\Gamma \vdash v : \emptyset \wedge v \notin \mathrm{Dom}(p))$