Reasonably Programmable Literal Notation
(ICFP 2018 Anonymous Supplemental Material)

March 16, 2018

Contents

Appendix A

Additional Examples

A.1 HTML Literals

Reason is seeing increasing adoption in the domain of client-side web programming, where generating and manipulating HTML data is central. There are many ways to represent HTML data in Reason, but one simple and useful representation is specified by `Html.t`, defined in Fig. A.1a. However, manually applying the constructors of this type is tedious and, for many web programmers, the induced notation is unfamiliar. It also makes it difficult to copy-and-paste from, for example, existing HTML files, as might be useful when refactoring an existing project to use Reason. Instead, we would like HTML literal notation. The `$html` TLM defined in Fig. A.1b implements the standard HTML notation extended with the tags `<$>` and `<$$>`, which support `Html.t` splicing and **`string`** splicing, respectively. It also supports the suffix ... on a tag, which causes the body of the tag to be a splice of type `list(Html.t)`.[1] The example in the paper demonstrated the first of these. The example in Fig. A.1d shows the list splicing form in pattern position.

In implementing HTML literals, using a parser generator may not be ideal. Instead, "post-processing" the result of an existing parser is a good approach. Fig. A.1c shows much of the implementation of `HtmlParser`, which is constructed by applying a functor that constructs a module that has the same essential interface as the modules that Menhir generates, with the functions `expr : string -> ProtoExpr.t` and `pat : string -> ProtoPat.t` of the input module exported as the "non-terminals". The companion lexer is `Relit.TrivLexer`. The implementation of `expr` on Line 23 of Fig. A.1c reveals that the literal body is parsed by first calling `ProtoHtml.parse`, then passing the result on to `ExprExpander.expand_proto`.

The `ProtoHtml.parse` function, outlined on Lines 8-9, generates a value of the type `ProtoHtml.t` defined on Lines 2-6 and so named because it is similar to `Html.t` but distinguishes spliced segments, like a proto-expression. We omit the definition of `ProtoHtml.parse` but conceptually, it starts by creating a character stream from the literal body, then from that creates a `Markup.ml` HTML content stream, then transforms that to a stream that emits the signals from the HTML content stream until it sees a splice start tag (which is reported as a syntax error relative to the HTML standard), at which point it calls `Segment.read_to`,

[1] We would also in practice have splicing for attribute values, but for simplicity we omit these.

which was described in the paper, to advance the original character stream until it recognizes the closing tag, emitting the generated segment and then returning control to the HTML content stream. The final step is to fold over this stream to produce the desired result of type `ProtoHtml.t`.

The `ExprExpander.expand_proto` function defined, with some helper functions, on Lines 9-20 finishes the job of the TLM parser by mapping the parsed proto-HTML to a proto-expression using the `$proto_expr` TLM previously described (in this case, using the `$list` antiquotation form that produces a parse tree of OCaml's built in list literal form from a list of parse trees). It would be straightforward to reuse the `ProtoHtml` module for different variations on HTML encodings, implementing only the expander anew. Overall, the amount of code needed to implement HTML literals.

Reason currently builds in "JSX" literals, which also support an HTML-like notation with support for splicing in a somewhat idiosynchratic manner motivated by certain JavaScript libraries (in particular, React). The expansion of this notation must be interpretted by a PPX rewriter. The problem is that this makes it difficult to use different HTML-related libraries at once. For example, ReasonReact and an XML library might both want to use this notation. This approach solves this problem (as well as other problems with the PPX-based approach as discussed in the paper).

A.2 Implementing Quasiquotation

The Reason quasiquotation TLMs, e.g. `$re_expr`, can similarly be implemented by the functional transformations outlined below, with an example from each step (grossly simplified from the actual `Parsetree` representation) on the right. In words, `$re_expr` first programmatically invokes the Reason parser on the literal body. It next serializes the generated parse tree to Reason source code, then parses that. This produces a parse tree that, if evaluated in the appropriate environment, will produce the original parse tree. The final step is to implement antiquotation as described above by repurposing the generalized literal forms in the body, using the source locations from the first parse tree, which have been carried into the second parse tree as constants.

```
body             "2 + `(xyz)`"
|> parse_re      Plus(Num(2, Loc(0)), GenLit("xyz", Loc(6)))
|> serialize_re  "Plus(Num(2, Loc(0)), GenLit(\"xyz\", Loc(6)))"
|> parse_re      Ap(V("Plus"), /*...*/Ap(V("GenLit"), Pair(Str("xyz"), /*...*/Num(6)))
|> genlit_to_sp  Ap(V("Plus"), /*...*/Spliced(6,8,TyPath(["ProtoExpr","t"])))
```

```
module Html = {
  /* a simplified encoding of HTML */
  type tag = string;
  type attr = (string, string);
  type attrs = list(attr);
  type t =
    | Text(string)
    | Elem(tag, attrs, list(t));
}
```

(a) The `Html` module, which defines `Html.t`.

```
module HtmlNotation {
  notation $html at Html.t {
    lexer Relit.TrivLexer;
    expression parser HtmlParser.expr;
    pattern parser HtmlParser.pat;
    expansions require
      { module Html = Html; }
  }
}
```

(b) The `$html` expression and pattern TLMs.

```
module HtmlParser = Relit.HandRolledExprPatParser({
  module ProtoHtml = {
    type t = PText(string)
           | PElem(Html.tag, Html.attrs, list(t))
           | PSplicedChildren(Html.tag, Html.attrs, Relit.Segment.t)
           | PSplicedElem(Relit.Segment.t)
           | PSplicedText(Relit.Segment.t)
    let parse : string => t =
      /* ... via Markup.ml's stream combinators (see text) ... */;
  };
  module ExprExpander = {
    open notation Relit.$proto_expr; open ProtoHtml;
    let expand_attr = (x1, x2) => `( ($s `(x1)`, $s `(x2)`) )`;
    let expand_attrs = (attrs) => `( $list `(List.map(expand_attr, attrs))` )`;
    let expand_proto = fun
    | PText(text) => `( Html.Text($s `(text)`) )`
    | PElem(tag, attrs, children) =>
      `( Html.Elem($s `(tag)`, `(expand_attrs(attrs))`,
          $list `(List.map(expand_parsed, children))`) )`
    | PSplicedChildren(tag, attrs, seg) =>
      `( Html.Elem($s `(tag)`, `(expand_attrs(attrs))`,
          $spliced `(seg : list(Html.t))`) )`
    | PSplicedHtml(seg) => `( $spliced `(seg : Html.t) )`);
    | PSplicedText(seg) => `( Html.Text($spliced `(seg : string)`) )`
  };
  let expr = (body) => body |> ProtoHtml.parse |> ExprExpander.expand_proto;
  let pat  = (body) => expr(body) |> ProtoPat.from_expr;
})
```

(c) The implementation of `HtmlParser`, which defers the parsing step to the `Markup.ml` library [1].

```
open HtmlNotation;
exception UnexpectedFormat;
let scrape_data = $html.(fun
| `(<div... class="result">
    [ `(<h1><$$>title</$$></h1>)`, author, summary, ..._ ]</div>)` =>
  (title, author, summary)
| _ => raise UnexpectedFormat);
```

(d) Using a pattern TLM to deconstruct an HTML value.

Figure A.1: Case Study: HTML literals as a library

Appendix B

$\mathsf{ML}^{\mathsf{Lit}}$: A Calculus of Simple TLMs

This section defines $\mathsf{ML}^{\mathsf{Lit}}$, the calculus of simple expression and pattern TLMs. For some readers, it might be useful to snip out pattern matching to get a language strictly of expression TLMs. To support that, one can omit the segments typeset in gray backgrounds below to recover $\mathsf{ML}^{\mathsf{ELit}}$, a calculus of simple expression TLMs. We have included the necessary eliminators below (they are technically redundant with pattern matching, but don't hurt things so they're left in white.)

B.1 Typographic Conventions

We adopt *PFPL*'s typographic conventions for abstract binding trees [2]. In particular, the names of operators and indexed families of operators are written in `typewriter font`, indexed families of operators specify indices within [braces] (except when the index is a label set, L, or natural number, n, in which case it is omitted). Term arguments are grouped roughly by sort using `{`curly braces`}` and `(`rounded braces`)`. We write $p.e$ for expressions binding the variables that appear in the pattern p. The variables in a pattern must be distinct.

We write $\{i \hookrightarrow \tau_i\}_{i \in L}$ for an unordered collection of type arguments τ_i, one for each $i \in L$, and similarly for arguments of other sorts. Similarly, we write $\{i \hookrightarrow J_i\}_{i \in L}$ for the finite set of derivations J_i for each $i \in L$.

We write $\{r_i\}_{1 \leq i \leq n}$ for sequences of $n \geq 0$ rule arguments, and similarly for other finite sequences.

Empty finite sets and finite functions are written $\emptyset$, or omitted entirely within judgements, and non-empty finite sets and finite functions are written as comma-separated sequences identified up to exchange and contraction.

B.2 Core Language

B.2.1 Syntax

Sort			Operational Form	Description
Typ	τ	$::=$	t	variable
			$\mathtt{parr}(\tau;\tau)$	partial function
			$\mathtt{all}(t.\tau)$	polymorphic
			$\mathtt{rec}(t.\tau)$	recursive
			$\mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i\in L})$	labeled product
			$\mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L})$	labeled sum
Exp	e	$::=$	x	variable
			$\mathtt{lam}\{\tau\}(x.e)$	abstraction
			$\mathtt{ap}(e;e)$	application
			$\mathtt{tlam}(t.e)$	type abstraction
			$\mathtt{tap}\{\tau\}(e)$	type application
			$\mathtt{fold}(e)$	fold
			$\mathtt{unfold}(e)$	unfold
			$\mathtt{tpl}(\{i \hookrightarrow e_i\}_{i\in L})$	labeled tuple
			$\mathtt{prj}[\ell](e)$	projection
			$\mathtt{inj}[\ell](e)$	injection
			$\mathtt{case}(e;\{i \hookrightarrow x_i.e_i\}_{i\in L})$	case analysis
			$\mathtt{match}(e;\{r_i\}_{1\leq i\leq n})$	match
Rule	r	$::=$	$\mathtt{rule}(p.e)$	rule
Pat	p	$::=$	x	variable pattern
			$\mathtt{wildp}$	wildcard pattern
			$\mathtt{foldp}(p)$	fold pattern
			$\mathtt{tplp}(\{i \hookrightarrow p_i\}_{i\in L})$	labeled tuple pattern
			$\mathtt{injp}[\ell](p)$	injection pattern

B.2.2 Static Semantics

Type formation contexts, Δ, are finite sets of hypotheses of the form t type. We write Δ, t type when t type $\notin \Delta$ for Δ extended with the hypothesis t type.

Typing contexts, Γ, are finite functions that map each variable $x \in \text{dom}(\Gamma)$, where $\text{dom}(\Gamma)$ is a finite set of variables, to the hypothesis $x : \tau$, for some τ. We write $\Gamma, x : \tau$, when $x \notin \text{dom}(\Gamma)$, for the extension of Γ with a mapping from x to $x : \tau$, and $\Gamma \cup \Gamma'$ when $\text{dom}(\Gamma) \cap \text{dom}(\Gamma') = \emptyset$ for the typing context mapping each $x \in \text{dom}(\Gamma) \cup \text{dom}(\Gamma')$ to $x : \tau$ if $x : \tau \in \Gamma$ or $x : \tau \in \Gamma'$. We write $\Delta \vdash \Gamma$ ctx if every type in Γ is well-formed relative to Δ.

Definition B.1 (Typing Context Formation)**.** $\Delta \vdash \Gamma$ ctx *iff for each hypothesis* $x : \tau \in \Gamma$, *we have* $\Delta \vdash \tau$ type.

$\boxed{\Delta \vdash \tau \;\mathsf{type}}$ τ is a well-formed type

$$\frac{}{\Delta, t \;\mathsf{type} \vdash t \;\mathsf{type}} \tag{B.1a}$$

$$\frac{\Delta \vdash \tau_1 \;\mathsf{type} \qquad \Delta \vdash \tau_2 \;\mathsf{type}}{\Delta \vdash \mathtt{parr}(\tau_1; \tau_2) \;\mathsf{type}} \tag{B.1b}$$

$$\frac{\Delta, t \;\mathsf{type} \vdash \tau \;\mathsf{type}}{\Delta \vdash \mathtt{all}(t.\tau) \;\mathsf{type}} \tag{B.1c}$$

$$\frac{\Delta, t \;\mathsf{type} \vdash \tau \;\mathsf{type}}{\Delta \vdash \mathtt{rec}(t.\tau) \;\mathsf{type}} \tag{B.1d}$$

$$\frac{\{\Delta \vdash \tau_i \;\mathsf{type}\}_{i \in L}}{\Delta \vdash \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}) \;\mathsf{type}} \tag{B.1e}$$

$$\frac{\{\Delta \vdash \tau_i \;\mathsf{type}\}_{i \in L}}{\Delta \vdash \mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}) \;\mathsf{type}} \tag{B.1f}$$

$\boxed{\Delta\; \Gamma \vdash e : \tau}$ e is assigned type τ

$$\frac{}{\Delta\; \Gamma, x : \tau \vdash x : \tau} \tag{B.2a}$$

$$\frac{\Delta \vdash \tau \;\mathsf{type} \qquad \Delta\; \Gamma, x : \tau \vdash e : \tau'}{\Delta\; \Gamma \vdash \mathtt{lam}\{\tau\}(x.e) : \mathtt{parr}(\tau; \tau')} \tag{B.2b}$$

$$\frac{\Delta\; \Gamma \vdash e_1 : \mathtt{parr}(\tau; \tau') \qquad \Delta\; \Gamma \vdash e_2 : \tau}{\Delta\; \Gamma \vdash \mathtt{ap}(e_1; e_2) : \tau'} \tag{B.2c}$$

$$\frac{\Delta, t \;\mathsf{type}\; \Gamma \vdash e : \tau}{\Delta\; \Gamma \vdash \mathtt{tlam}(t.e) : \mathtt{all}(t.\tau)} \tag{B.2d}$$

$$\frac{\Delta\; \Gamma \vdash e : \mathtt{all}(t.\tau) \qquad \Delta \vdash \tau' \;\mathsf{type}}{\Delta\; \Gamma \vdash \mathtt{tap}\{\tau'\}(e) : [\tau'/t]\tau} \tag{B.2e}$$

$$\frac{\Delta\; \Gamma \vdash e : [\mathtt{rec}(t.\tau)/t]\tau}{\Delta\; \Gamma \vdash \mathtt{fold}(e) : \mathtt{rec}(t.\tau)} \tag{B.2f}$$

$$\frac{\Delta\; \Gamma \vdash e : \mathtt{rec}(t.\tau)}{\Delta\; \Gamma \vdash \mathtt{unfold}(e) : [\mathtt{rec}(t.\tau)/t]\tau} \tag{B.2g}$$

$$\frac{\{\Delta\; \Gamma \vdash e_i : \tau_i\}_{i \in L}}{\Delta\; \Gamma \vdash \mathtt{tpl}(\{i \hookrightarrow e_i\}_{i \in L}) : \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L})} \tag{B.2h}$$

$$\frac{\Delta\;\Gamma \vdash e : \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i\in L}; \ell \hookrightarrow \tau)}{\Delta\;\Gamma \vdash \texttt{prj}[\ell](e) : \tau} \tag{B.2i}$$

$$\frac{\Delta\;\Gamma \vdash e : \tau}{\Delta\;\Gamma \vdash \texttt{inj}[\ell](e) : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L}; \ell \hookrightarrow \tau)} \tag{B.2j}$$

$$\frac{\Delta\;\Gamma \vdash e : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L}) \qquad \{\Delta\;\Gamma, x_i : \tau_i \vdash e_i : \tau\}_{i\in L}}{\Delta\;\Gamma \vdash \texttt{case}(e; \{i \hookrightarrow x_i.e_i\}_{i\in L}) : \tau} \tag{B.2k}$$

$$\frac{\Delta\;\Gamma \vdash e : \tau \qquad \{\Delta\;\Gamma \vdash r_i : \tau \Rrightarrow \tau'\}_{1\le i\le n}}{\Delta\;\Gamma \vdash \texttt{match}(e; \{r_i\}_{1\le i\le n}) : \tau'} \tag{B.2l}$$

$\boxed{\Delta\;\Gamma \vdash r : \tau \Rrightarrow \tau'}$ r takes values of type τ to values of type τ'

$$\frac{p : \tau \dashV \Gamma' \qquad \Delta\;\Gamma \cup \Gamma' \vdash e : \tau'}{\Delta\;\Gamma \vdash \texttt{rule}(p.e) : \tau \Rrightarrow \tau'} \tag{B.3}$$

Rule (B.3) is defined mutually inductively with Rules (B.2).

$\boxed{p : \tau \dashV \Gamma}$ p matches values of type τ and generates hypotheses Γ

$$\frac{}{x : \tau \dashV x : \tau} \tag{B.4a}$$

$$\frac{}{\texttt{wildp} : \tau \dashV \emptyset} \tag{B.4b}$$

$$\frac{p : [\texttt{rec}(t.\tau)/t]\tau \dashV \Gamma}{\texttt{foldp}(p) : \texttt{rec}(t.\tau) \dashV \Gamma} \tag{B.4c}$$

$$\frac{\{p_i : \tau_i \dashV \Gamma_i\}_{i\in L}}{\texttt{tplp}(\{i \hookrightarrow p_i\}_{i\in L}) : \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i\in L}) \dashV \cup_{i\in L}\Gamma_i} \tag{B.4d}$$

$$\frac{p : \tau \dashV \Gamma}{\texttt{injp}[\ell](p) : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L}; \ell \hookrightarrow \tau) \dashV \Gamma} \tag{B.4e}$$

Metatheory

The rules above are syntax-directed, so we assume an inversion lemma for each rule as needed without stating it separately or proving it explicitly. The following standard lemmas also hold.

The Weakening Lemma establishes that extending the context with unnecessary hypotheses preserves well-formedness and typing.

Lemma B.2 (Weakening).

1. *If* $\Delta \vdash \tau$ type *then* Δ, t type $\vdash \tau$ type.
2. (a) *If* $\Delta\ \Gamma \vdash e : \tau$ *then* Δ, t type $\Gamma \vdash e : \tau$.
   (b) *If* $\Delta\ \Gamma \vdash r : \tau \Mapsto \tau'$ *then* Δ, t type $\Gamma \vdash r : \tau \Mapsto \tau'$.
3. (a) *If* $\Delta\ \Gamma \vdash e : \tau$ *and* $\Delta \vdash \tau''$ type *then* $\Delta\ \Gamma, x : \tau'' \vdash e : \tau$.
   (b) *If* $\Delta\ \Gamma \vdash r : \tau \Mapsto \tau'$ *and* $\Delta \vdash \tau''$ type *then* $\Delta\ \Gamma, x : \tau'' \vdash r : \tau \Mapsto \tau'$.
4. *If* $p : \tau \dashv\!\!\mid \Gamma$ *then* Δ, t type $\vdash p : \tau \dashv\!\!\mid \Gamma$.

Proof Sketch.

1. By rule induction over Rules (B.1).
2. By mutual rule induction over Rules (B.2) and Rule (B.3), and part 1.
3. By mutual rule induction over Rules (B.2) and Rule (B.3), and part 1.
4. By rule induction over Rules (B.4).

□

Note clause 4, which allows weakening of Δ but requires that the pattern typing judgement is *linear* in the pattern typing context, i.e. it does *not* obey weakening of the pattern typing context. This is to ensure that the pattern typing context captures exactly those hypotheses generated by a pattern, and no others.

The Substitution Lemma establishes that substitution of a well-formed type for a type variable, or an expanded expression of the appropriate type for an expanded expression variable, preserves well-formedness and typing.

Lemma B.3 (Substitution).

1. *If* Δ, t type $\vdash \tau$ type *and* $\Delta \vdash \tau'$ type *then* $\Delta \vdash [\tau'/t]\tau$ type.
2. (a) *If* Δ, t type $\Gamma \vdash e : \tau$ *and* $\Delta \vdash \tau'$ type *then* $\Delta\ [\tau'/t]\Gamma \vdash [\tau'/t]e : [\tau'/t]\tau$.
   (b) *If* Δ, t type $\Gamma \vdash r : \tau \Mapsto \tau''$ *and* $\Delta \vdash \tau'$ type *then* $\Delta\ [\tau'/t]\Gamma \vdash [\tau'/t]r : [\tau'/t]\tau \Mapsto [\tau'/t]\tau''$.
3. (a) *If* $\Delta\ \Gamma, x : \tau' \vdash e : \tau$ *and* $\Delta\ \Gamma \vdash e' : \tau'$ *then* $\Delta\ \Gamma \vdash [e'/x]e : \tau$.
   (b) *If* $\Delta\ \Gamma, x : \tau' \vdash r : \tau \Mapsto \tau''$ *and* $\Delta\ \Gamma \vdash e' : \tau''$ *then* $\Delta\ \Gamma \vdash [e'/x]r : \tau \Mapsto \tau''$.

Proof Sketch.

1. By rule induction over Rules (B.1).
2. By mutual rule induction over Rules (B.2) and Rule (B.3).

3. By mutual rule induction over Rules (B.2) and Rule (B.3).

□

The Decomposition Lemma is the converse of the Substitution Lemma.

Lemma B.4 (Decomposition)**.**

1. *If* $\Delta \vdash [\tau'/t]\tau$ type *and* $\Delta \vdash \tau'$ type *then* Δ, t type $\vdash \tau$ type.
2. (a) *If* $\Delta\ [\tau'/t]\Gamma \vdash [\tau'/t]e : [\tau'/t]\tau$ *and* $\Delta \vdash \tau'$ type *then* Δ, t type $\Gamma \vdash e : \tau$.
   (b) *If* $\Delta\ [\tau'/t]\Gamma \vdash [\tau'/t]r : [\tau'/t]\tau \mapsto [\tau'/t]\tau''$ *and* $\Delta \vdash \tau'$ type *then* Δ, t type $\Gamma \vdash r : \tau \mapsto \tau''$.
3. (a) *If* $\Delta\ \Gamma \vdash [e'/x]e : \tau$ *and* $\Delta\ \Gamma \vdash e' : \tau'$ *then* $\Delta\ \Gamma, x : \tau' \vdash e : \tau$.
   (b) *If* $\Delta\ \Gamma \vdash [e'/x]r : \tau \mapsto \tau''$ *and* $\Delta\ \Gamma \vdash e' : \tau'$ *then* $\Delta\ \Gamma, x : \tau' \vdash r : \tau \mapsto \tau''$.

Proof Sketch.

1. By rule induction over Rules (B.1) and case analysis over the definition of substitution. In all cases, the derivation of $\Delta \vdash [\tau'/t]\tau$ type does not depend on the form of τ'.
2. By mutual rule induction over Rules (B.2) and Rule (B.3) and case analysis over the definition of substitution. In all cases, the derivation of $\Delta\ [\tau'/t]\Gamma \vdash [\tau'/t]e : [\tau'/t]\tau$ or $\Delta\ [\tau'/t]\Gamma \vdash [\tau'/t]r : [\tau'/t]\tau \mapsto [\tau'/t]\tau''$ does not depend on the form of τ'.
3. By mutual rule induction over Rules (B.2) and Rule (B.3) and case analysis over the definition of substitution. In all cases, the derivation of $\Delta\ \Gamma \vdash [e'/x]e : \tau$ or $\Delta\ \Gamma \vdash [e'/x]r : \tau \mapsto \tau''$ does not depend on the form of e'.

□

Lemma B.5 (Pattern Regularity)**.** *If* $p : \tau \dashv\vdash \Gamma$ *and* $\Delta \vdash \tau$ type *then* $\Delta \vdash \Gamma$ ctx *and* $\mathsf{patvars}(p) = dom(\Gamma)$.

Proof. By rule induction over Rules (B.4).

Case (B.4a)**.**

(1)	$p = x$	by assumption
(2)	$\Gamma = x : \tau$	by assumption
(3)	$\Delta \vdash \tau$ type	by assumption
(4)	$\Delta \vdash x : \tau$ ctx	by Definition B.1 on (3)
(5)	$\mathsf{fv}(p) = \mathrm{dom}(\Gamma) = \{x\}$	by definition

Case (B.4b)**.**

(1)	$p = \texttt{wildp}$	by assumption
(2)	$\Gamma = \emptyset$	by assumption
(3)	$\Delta \vdash \emptyset \; \mathsf{ctx}$	by Definition B.1
(4)	$\mathsf{patvars}(p) = \mathrm{dom}(\Gamma) = \emptyset$	by definition

Case (B.4d).

(1)	$p = \texttt{tplp}(\{i \hookrightarrow p_i\}_{i \in L})$	by assumption
(2)	$\tau = \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L})$	by assumption
(3)	$\Gamma = \cup_{i \in L} \Gamma_i$	by assumption
(4)	$\{p_i : \tau_i \dashv\!\vert\, \Gamma_i\}_{i \in L}$	by assumption
(5)	$\Delta \vdash \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}) \; \mathsf{type}$	by assumption
(6)	$\{\Delta \vdash \tau_i \; \mathsf{type}\}_{i \in L}$	by Inversion of Rule (B.1e) on (5)
(7)	$\{\Delta \vdash \Gamma_i \; \mathsf{ctx}\}_{i \in L}$	by IH over (4) and (6)
(8)	$\{\mathsf{patvars}(p_i) = \mathrm{dom}(\Gamma_i)\}_{i \in L}$	by IH over (4) and (6)
(9)	$\Delta \vdash \cup_{i \in L} \Gamma_i \; \mathsf{ctx}$	by Definition B.1 over (7), then Definition B.1 iteratively
(10)	$\mathsf{patvars}(p) = \mathrm{dom}(\Gamma) = \emptyset$	by definition and (8)

Case (B.4e).

(1)	$p = \texttt{injp}[\ell](p')$	by assumption
(2)	$\tau = \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau')$	by assumption
(3)	$\Delta \vdash \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau') \; \mathsf{type}$	by assumption
(4)	$p' : \tau' \dashv\!\vert\, \Gamma$	by assumption
(5)	$\Delta \vdash \tau' \; \mathsf{type}$	by Inversion of Rule (B.1f) on (3)
(6)	$\Delta \vdash \Gamma \; \mathsf{ctx}$	by IH on (4) and (5)
(7)	$\mathsf{patvars}(p') = \mathrm{dom}(\Gamma)$	by IH on (4) and (5)
(8)	$\mathsf{patvars}(p) = \mathrm{dom}(\Gamma)$	by definition and (7)

$\square$

B.2.3 Structural Operational Semantics

The *structural operational semantics* is specified as a transition system, and is organized around judgements of the following form:

Judgement Form	**Description**
$e \mapsto e'$	e transitions to e'
e val	e is a value
e matchfail	e raises match failure

We also define auxiliary judgements for *iterated transition*, $e \mapsto^* e'$, and *evaluation*, $e \Downarrow e'$.

Definition B.6 (Iterated Transition). *Iterated transition, $e \mapsto^* e'$, is the reflexive, transitive closure of the transition judgement, $e \mapsto e'$.*

Definition B.7 (Evaluation). *$e \Downarrow e'$ iff $e \mapsto^* e'$ and e'* val.

Our subsequent developments do not make mention of particular rules in the dynamic semantics, nor do they make mention of other judgements, not listed above, that are used only for defining the dynamics of the match operator, so we do not produce these details here. Instead, it suffices to state the following conditions.

Condition B.8 (Canonical Forms). *If $\vdash e : \tau$ and e* val *then:*

1. *If $\tau = \texttt{parr(}\tau_1\texttt{;}\tau_2\texttt{)}$ then $e = \texttt{lam\{}\tau_1\texttt{\}(}x.e'\texttt{)}$ and $x : \tau_1 \vdash e' : \tau_2$.*

2. *If $\tau = \texttt{all(}t.\tau'\texttt{)}$ then $e = \texttt{tlam(}t.e'\texttt{)}$ and t* type *$\vdash e' : \tau'$.*

3. *If $\tau = \texttt{rec(}t.\tau'\texttt{)}$ then $e = \texttt{fold(}e'\texttt{)}$ and $\vdash e' : [\texttt{rec(}t.\tau'\texttt{)}/t]\tau'$ and e'* val.

4. *If $\tau = \texttt{prod(}\{i \hookrightarrow \tau_i\}_{i \in L}\texttt{)}$ then $e = \texttt{tpl(}\{i \hookrightarrow e_i\}_{i \in L}\texttt{)}$ and $\vdash e_i : \tau_i$ and e_i* val *for each $i \in L$.*

5. *If $\tau = \texttt{sum(}\{i \hookrightarrow \tau_i\}_{i \in L}\texttt{)}$ then for some label set L' and label ℓ and type τ', we have that $L = L', \ell$ and $\tau = \texttt{sum(}\{i \hookrightarrow \tau_i\}_{i \in L'}\texttt{;}\ell \hookrightarrow \tau'\texttt{)}$ and $e = \texttt{inj}[\ell]\texttt{(}e'\texttt{)}$ and $\vdash e' : \tau'$ and e'* val.

Condition B.9 (Preservation). *If $\vdash e : \tau$ and $e \mapsto e'$ then $\vdash e' : \tau$.*

Condition B.10 (Progress). *If $\vdash e : \tau$ then either e* val *or e* matchfail *or there exists an e' such that $e \mapsto e'$.*

B.3 Unexpanded Language (UL)

B.3.1 Syntax

Stylized Syntax

Sort			**Stylized Form**	**Description**
UTyp	$\hat{\tau}$	::=	$\hat{t}$	identifier
			$\hat{\tau} \rightharpoonup \hat{\tau}$	partial function
			$\forall \hat{t}.\hat{\tau}$	polymorphic
			$\mu \hat{t}.\hat{\tau}$	recursive
			$\langle \{i \hookrightarrow \hat{\tau}_i\}_{i \in L} \rangle$	labeled product
			$[\{i \hookrightarrow \hat{\tau}_i\}_{i \in L}]$	labeled sum
UExp	$\hat{e}$	::=	$\hat{x}$	identifier
			$\hat{e} : \hat{\tau}$	ascription
			$\texttt{let val}\ \hat{x} = \hat{e}\ \texttt{in}\ \hat{e}$	value binding
			$\lambda \hat{x}{:}\hat{\tau}.\hat{e}$	abstraction
			$\hat{e}(\hat{e})$	application
			$\Lambda \hat{t}.\hat{e}$	type abstraction
			$\hat{e}[\hat{\tau}]$	type application
			$\texttt{fold}(\hat{e})$	fold
			$\texttt{unfold}(\hat{e})$	unfold
			$\langle \{i \hookrightarrow \hat{e}_i\}_{i \in L} \rangle$	labeled tuple
			$\hat{e} \cdot \ell$	projection
			$\texttt{inj}[\ell](\hat{e})$	injection
			$\texttt{case}\ \hat{e}\ \{i \hookrightarrow \hat{x}_i.\hat{e}_i\}_{i \in L}$	case analysis
			$\texttt{notation}\ \hat{a}\ \texttt{at}\ \hat{\tau}$	
			$\{\ \texttt{expr parser}\ e; \texttt{expansions require}\ \hat{e}\ \}\ \texttt{in}\ \hat{e}$	seTLM definition
			$\hat{a}$ ‘(b)‘	seTLM application
			$\texttt{match}\ \hat{e}\ \{\hat{r}_i\}_{1 \le i \le n}$	match
			$\texttt{notation}\ \hat{a}\ \texttt{at}\ \hat{\tau}\ \{\ \texttt{pat parser}\ e\ \}\ \texttt{in}\ \hat{e}$	spTLM definition
URule	$\hat{r}$	::=	$\hat{p} \Rightarrow \hat{e}$	match rule
UPat	$\hat{p}$	::=	$\hat{x}$	identifier pattern
			_	wildcard pattern
			$\texttt{fold}(\hat{p})$	fold pattern
			$\langle \{i \hookrightarrow \hat{p}_i\}_{i \in L} \rangle$	labeled tuple pattern
			$\texttt{inj}[\ell](\hat{p})$	injection pattern
			$\hat{a}$ ‘(b)‘	spTLM application

Body Lengths We write $\|b\|$ for the length of b. The metafunction $\|\hat{e}\|$ computes the sum of the lengths of expression literal bodies in $\hat{e}$:

$$
\begin{array}{ll}
\|\hat{x}\| & = 0 \\
\|\hat{e} : \hat{\tau}\| & = \|\hat{e}\| \\
\|\texttt{let}\,\texttt{val}\ \hat{x} = \hat{e}_1\ \texttt{in}\ \hat{e}_2\| & = \|\hat{e}_1\| + \|\hat{e}_2\| \\
\|\lambda\hat{x}{:}\hat{\tau}.\hat{e}\| & = \|\hat{e}\| \\
\|\hat{e}_1(\hat{e}_2)\| & = \|\hat{e}_1\| + \|\hat{e}_2\| \\
\|\Lambda\hat{t}.\hat{e}\| & = \|\hat{e}\| \\
\|\hat{e}\texttt{[}\hat{\tau}\texttt{]}\| & = \|\hat{e}\| \\
\|\texttt{fold}(\hat{e})\| & = \|\hat{e}\| \\
\|\texttt{unfold}(\hat{e})\| & = \|\hat{e}\| \\
\|\langle\{i \hookrightarrow \hat{e}_i\}_{i\in L}\rangle\| & = \sum_{i\in L}\|\hat{e}_i\| \\
\|\ell \cdot \hat{e}\| & = \|\hat{e}\| \\
\|\texttt{inj}[\ell](\hat{e})\| & = \|\hat{e}\| \\
\|\texttt{case}\ \hat{e}\ \{i \hookrightarrow \hat{x}_i.\hat{e}_i\}_{i\in L}\| & = \|\hat{e}\| + \sum_{i\in L}\|\hat{e}_i\| \\
\|\texttt{notation}\ \hat{a}\ \texttt{at}\ \hat{\tau}\ \{\ \texttt{expr}\ \texttt{parser}\ e; \texttt{expansions}\ \texttt{require}\ \hat{e}\ \}\ \texttt{in}\ \hat{e}'\| & = \|\hat{e}\| + \|\hat{e}'\| \\
\|\hat{a}\ \texttt{`(}b\texttt{)`}\| & = \|b\| \\
\|\texttt{match}\ \hat{e}\ \{\hat{r}_i\}_{1\le i\le n}\| & = \|\hat{e}\| + \sum_{1\le i\le n}\|r_i\| \\
\|\texttt{notation}\ \hat{a}\ \texttt{at}\ \hat{\tau}\ \{\ \texttt{pat}\ \texttt{parser}\ e\ \}\ \texttt{in}\ \hat{e}\| & = \|\hat{e}\|
\end{array}
$$

and $\|\hat{r}\|$ computes the sum of the lengths of expression literal bodies in $\hat{r}$:

$$\|\hat{p} \Rightarrow \hat{e}\| = \|\hat{e}\|$$

Similarly, the metafunction $\|\hat{p}\|$ computes the sum of the lengths of the pattern literal bodies in $\hat{p}$:

$$
\begin{aligned}
\|\hat{x}\| &= 0 \\
\|\texttt{fold(}\hat{p}\texttt{)}\| &= \|\hat{p}\| \\
\|\langle\{i \hookrightarrow \hat{p}_i\}_{i\in L}\rangle\| &= \sum_{i\in L}\|\hat{p}_i\| \\
\|\texttt{inj}[\ell](\hat{p})\| &= \|\hat{p}\| \\
\|\hat{a}\ \texttt{`(}b\texttt{)`}\| &= \|b\|
\end{aligned}
$$

Common Unexpanded Forms Each expanded form maps onto an unexpanded form. We refer to these as the *common forms*. In particular:

- Each type variable, t, maps onto a unique type identifier, written $\widehat{t}$.

- Each type, τ, maps onto an unexpanded type, $\mathcal{U}(\tau)$, as follows:

$$\begin{aligned}
\mathcal{U}(t) &= \widehat{t} \\
\mathcal{U}(\texttt{parr}(\tau_1;\tau_2)) &= \mathcal{U}(\tau_1) \rightharpoonup \mathcal{U}(\tau_2) \\
\mathcal{U}(\texttt{all}(t.\tau)) &= \forall\widehat{t}.\mathcal{U}(\tau) \\
\mathcal{U}(\texttt{rec}(t.\tau)) &= \mu\widehat{t}.\mathcal{U}(\tau) \\
\mathcal{U}(\texttt{prod}(\{i \hookrightarrow \tau_i\}_{i\in L})) &= \langle\{i \hookrightarrow \mathcal{U}(\tau_i)\}_{i\in L}\rangle \\
\mathcal{U}(\texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L})) &= [\{i \hookrightarrow \mathcal{U}(\tau_i)\}_{i\in L}]
\end{aligned}$$

- Each expression variable, x, maps onto a unique expression identifier, written $\widehat{x}$.

- Each core language expression, e, maps onto an unexpanded expression, $\mathcal{U}(e)$, as follows:

$$\begin{aligned}
\mathcal{U}(x) &= \widehat{x} \\
\mathcal{U}(\texttt{lam}\{\tau\}(x.e)) &= \lambda\widehat{x}{:}\mathcal{U}(\tau).\mathcal{U}(e) \\
\mathcal{U}(\texttt{ap}(e_1;e_2)) &= \mathcal{U}(e_1)(\mathcal{U}(e_2)) \\
\mathcal{U}(\texttt{tlam}(t.e)) &= \Lambda\widehat{t}.\mathcal{U}(e) \\
\mathcal{U}(\texttt{tap}\{\tau\}(e)) &= \mathcal{U}(e)\texttt{[}\mathcal{U}(\tau)\texttt{]} \\
\mathcal{U}(\texttt{fold}(e)) &= \texttt{fold}(\mathcal{U}(e)) \\
\mathcal{U}(\texttt{unfold}(e)) &= \texttt{unfold}(\mathcal{U}(e)) \\
\mathcal{U}(\texttt{tpl}(\{i \hookrightarrow e_i\}_{i\in L})) &= \langle\{i \hookrightarrow \mathcal{U}(e_i)\}_{i\in L}\rangle \\
\mathcal{U}(\texttt{prj}[\ell](e)) &= \mathcal{U}(e)\cdot\ell \\
\mathcal{U}(\texttt{inj}[\ell](e)) &= \texttt{inj}[\ell](\mathcal{U}(e)) \\
\mathcal{U}(\texttt{match}(e;\{r_i\}_{1\leq i\leq n})) &= \texttt{match}\,\mathcal{U}(e)\ \{\mathcal{U}(r_i)\}_{1\leq i\leq n}
\end{aligned}$$

- Each core language rule, r, maps onto an unexpanded rule, $\mathcal{U}(r)$, as follows:

$$\mathcal{U}(\texttt{rule}(p.e)) = \texttt{urule}(\mathcal{U}(p).\mathcal{U}(e))$$

- Each core language pattern, p, maps onto the unexpanded pattern, $\mathcal{U}(p)$, as follows:

$$\begin{aligned}
\mathcal{U}(x) &= \widehat{x} \\
\mathcal{U}(\texttt{wildp}) &= \texttt{uwildp} \\
\mathcal{U}(\texttt{foldp}(p)) &= \texttt{ufoldp}(\mathcal{U}(p)) \\
\mathcal{U}(\texttt{tplp}(\{i \hookrightarrow p_i\}_{i\in L})) &= \texttt{utplp}[L](\{i \hookrightarrow \mathcal{U}(p_i)\}_{i\in L}) \\
\mathcal{U}(\texttt{injp}[\ell](p)) &= \texttt{uinjp}[\ell](\mathcal{U}(p))
\end{aligned}$$

Textual Syntax

In addition to the stylized syntax, there is also a context-free textual syntax for the UL. For our purposes, we need only posit the existence of partial metafunctions $\mathsf{parseUTyp}(b)$ and $\mathsf{parseUExp}(b)$ and $\mathsf{parseUPat}(b)$.

Condition B.11 (Textual Representability)**.**

1. *For each* $\hat{\tau}$*, there exists* b *such that* $\mathsf{parseUTyp}(b) = \hat{\tau}$.

2. *For each* $\hat{e}$*, there exists* b *such that* $\mathsf{parseUExp}(b) = \hat{e}$.

3. *For each* $\hat{p}$*, there exists* b *such that* $\mathsf{parseUPat}(b) = \hat{p}$.

We also impose the following technical conditions.

Condition B.12 (Expression Parsing Monotonicity). *If* $\mathsf{parseUExp}(b) = \hat{e}$ *then* $\|\hat{e}\| < \|b\|$.

Condition B.13 (Pattern Parsing Monotonicity). *If* $\mathsf{parseUPat}(b) = \hat{p}$ *then* $\|\hat{p}\| < \|b\|$.

B.3.2 Type Expansion

Unexpanded type formation contexts, $\hat{\Delta}$, are of the form $\langle \mathcal{D}; \Delta \rangle$, i.e. they consist of a *type identifier expansion context*, $\mathcal{D}$, paired with a type formation context, Δ.

A *type identifier expansion context*, $\mathcal{D}$, is a finite function that maps each type identifier $\hat{t} \in \mathrm{dom}(\mathcal{D})$ to the hypothesis $\hat{t} \rightsquigarrow t$, for some type variable t. We write $\mathcal{D} \uplus \hat{t} \rightsquigarrow t$ for the type identifier expansion context that maps $\hat{t}$ to $\hat{t} \rightsquigarrow t$ and defers to $\mathcal{D}$ for all other type identifiers (i.e. the previous mapping is *updated*.)

We define $\hat{\Delta}, \hat{t} \rightsquigarrow t\ \mathsf{type}$ when $\hat{\Delta} = \langle \mathcal{D}; \Delta \rangle$ as an abbreviation of

$$\langle \mathcal{D} \uplus \hat{t} \rightsquigarrow t; \Delta, t\ \mathsf{type} \rangle$$

Definition B.14 (Unexpanded Type Formation Context Formation). $\vdash \langle \mathcal{D}; \Delta \rangle\ \mathsf{utctx}$ *iff for each* $\hat{t} \rightsquigarrow t\ \mathsf{type} \in \mathcal{D}$ *we have* $t\ \mathsf{type} \in \Delta$.

$\boxed{\hat{\Delta} \vdash \hat{\tau} \rightsquigarrow \tau\ \mathsf{type}}$ $\hat{\tau}$ has well-formed expansion τ

$$\frac{}{\hat{\Delta}, \hat{t} \rightsquigarrow t\ \mathsf{type} \vdash \hat{t} \rightsquigarrow t\ \mathsf{type}} \tag{B.5a}$$

$$\frac{\hat{\Delta} \vdash \hat{\tau}_1 \rightsquigarrow \tau_1\ \mathsf{type} \qquad \hat{\Delta} \vdash \hat{\tau}_2 \rightsquigarrow \tau_2\ \mathsf{type}}{\hat{\Delta} \vdash \mathtt{uparr}(\hat{\tau}_1; \hat{\tau}_2) \rightsquigarrow \mathtt{parr}(\tau_1; \tau_2)\ \mathsf{type}} \tag{B.5b}$$

$$\frac{\hat{\Delta}, \hat{t} \rightsquigarrow t\ \mathsf{type} \vdash \hat{\tau} \rightsquigarrow \tau\ \mathsf{type}}{\hat{\Delta} \vdash \mathtt{uall}(\hat{t}.\hat{\tau}) \rightsquigarrow \mathtt{all}(t.\tau)\ \mathsf{type}} \tag{B.5c}$$

$$\frac{\hat{\Delta}, \hat{t} \rightsquigarrow t\ \mathsf{type} \vdash \hat{\tau} \rightsquigarrow \tau\ \mathsf{type}}{\hat{\Delta} \vdash \mathtt{urec}(\hat{t}.\hat{\tau}) \rightsquigarrow \mathtt{rec}(t.\tau)\ \mathsf{type}} \tag{B.5d}$$

$$\frac{\{\hat{\Delta} \vdash \hat{\tau}_i \rightsquigarrow \tau_i\ \mathsf{type}\}_{i \in L}}{\hat{\Delta} \vdash \mathtt{uprod}[L](\{i \hookrightarrow \hat{\tau}_i\}_{i \in L}) \rightsquigarrow \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L})\ \mathsf{type}} \tag{B.5e}$$

$$\frac{\{\hat{\Delta} \vdash \hat{\tau}_i \rightsquigarrow \tau_i\ \mathsf{type}\}_{i \in L}}{\hat{\Delta} \vdash \mathtt{usum}[L](\{i \hookrightarrow \hat{\tau}_i\}_{i \in L}) \rightsquigarrow \mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L})\ \mathsf{type}} \tag{B.5f}$$

B.3.3 Typed Expression Expansion

Contexts

Unexpanded typing contexts, $\hat{\Gamma}$, are, similarly, of the form $\langle \mathcal{G}; \Gamma \rangle$, where $\mathcal{G}$ is an *expression identifier expansion context*, and Γ is a typing context. An expression identifier expansion context, $\mathcal{G}$, is a finite function that maps each expression identifier $\hat{x} \in \text{dom}(\mathcal{G})$ to the hypothesis $\hat{x} \rightsquigarrow x$, for some expression variable, x. We write $\mathcal{G} \uplus \hat{x} \rightsquigarrow x$ for the expression identifier expansion context that maps $\hat{x}$ to $\hat{x} \rightsquigarrow x$ and defers to $\mathcal{G}$ for all other expression identifiers (i.e. the previous mapping is updated.)

We define $\hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau$ when $\hat{\Gamma} = \langle \mathcal{G}; \Gamma \rangle$ as an abbreviation of

$$\langle \mathcal{G} \uplus \hat{x} \rightsquigarrow x; \Gamma, x : \tau \rangle$$

Definition B.15 (Unexpanded Typing Context Formation)**.** $\Delta \vdash \langle \mathcal{G}; \Gamma \rangle$ uctx *iff* $\Delta \vdash \Gamma$ ctx *and for each* $\hat{x} \rightsquigarrow x \in \mathcal{G}$*, we have* $x \in dom(\Gamma)$*.*

Body Encoding and Decoding

An assumed type abbreviated `Body` classifies encodings of literal bodies, b. The mapping from literal bodies to values of type `Body` is defined by the *body encoding judgement* $b \downarrow_{\mathsf{Body}} e_{\text{body}}$. An inverse mapping is defined by the *body decoding judgement* $e_{\text{body}} \uparrow_{\mathsf{Body}} b$.

Judgement Form	**Description**
$b \downarrow_{\mathsf{Body}} e$	b has encoding e
$e \uparrow_{\mathsf{Body}} b$	e has decoding b

The following condition establishes an isomorphism between literal bodies and values of type `Body` mediated by the judgements above.

Condition B.16 (Body Isomorphism)**.**

1. *For every literal body* b*, we have that* $b \downarrow_{\mathsf{Body}} e_{body}$ *for some* e_{body} *such that* $\vdash e_{body} : \texttt{Body}$ *and* e_{body} val.
2. *If* $\vdash e_{body} : \texttt{Body}$ *and* e_{body} val *then* $e_{body} \uparrow_{\mathsf{Body}} b$ *for some* b*.*
3. *If* $b \downarrow_{\mathsf{Body}} e_{body}$ *then* $e_{body} \uparrow_{\mathsf{Body}} b$*.*
4. *If* $\vdash e_{body} : \texttt{Body}$ *and* e_{body} val *and* $e_{body} \uparrow_{\mathsf{Body}} b$ *then* $b \downarrow_{\mathsf{Body}} e_{body}$*.*
5. *If* $b \downarrow_{\mathsf{Body}} e_{body}$ *and* $b \downarrow_{\mathsf{Body}} e'_{body}$ *then* $e_{body} = e'_{body}$*.*
6. *If* $\vdash e_{body} : \texttt{Body}$ *and* e_{body} val *and* $e_{body} \uparrow_{\mathsf{Body}} b$ *and* $e_{body} \uparrow_{\mathsf{Body}} b'$ *then* $b = b'$*.*

We also assume a partial metafunction, $\mathsf{subseq}(b; m; n)$, which extracts a subsequence of b starting at position m and ending at position n, inclusive, where m and n are natural numbers. The following condition is technically necessary.

Condition B.17 (Body Subsequencing)**.** *If* $\mathsf{subseq}(b; m; n) = b'$ *then* $\|b'\| \leq \|b\|$*.*

Parse Results

The type abbreviated ParseResultE, and an auxiliary abbreviation used below, is defined as follows:

$$L_{\mathtt{SE}} \stackrel{\text{def}}{=} \mathtt{Error}, \mathtt{SuccessE}$$

$$\mathtt{ParseResultE} \stackrel{\text{def}}{=} \mathtt{sum}(\mathtt{Error} \hookrightarrow \langle\rangle, \mathtt{SuccessE} \hookrightarrow \mathtt{PrExpr})$$

The type abbreviated ParseResultP, and an auxiliary abbreviation used below, is defined as follows:

$$L_{\mathtt{SP}} \stackrel{\text{def}}{=} \mathtt{Error}, \mathtt{SuccessP}$$

$$\mathtt{ParseResultE} \stackrel{\text{def}}{=} \mathtt{sum}(\mathtt{Error} \hookrightarrow \langle\rangle, \mathtt{SuccessP} \hookrightarrow \mathtt{PrPat})$$

seTLM Contexts

seTLM contexts, $\hat{\Psi}$, are of the form $\langle \mathcal{A}; \Psi \rangle$, where $\mathcal{A}$ is a *TLM identifier expansion context* and Ψ is a *seTLM definition context*.

A *TLM identifier expansion context*, $\mathcal{A}$, is a finite function mapping each TLM identifier $\hat{a} \in \text{dom}(\mathcal{A})$ to the *TLM identifier expansion*, $\hat{a} \rightsquigarrow x$, for some variable x. We write $\mathcal{A} \uplus \hat{a} \rightsquigarrow x$ for the TLM identifier expansion context that maps $\hat{a}$ to $\hat{a} \rightsquigarrow x$, and defers to $\mathcal{A}$ for all other TLM identifiers (i.e. the previous mapping is *updated*.)

An *seTLM definition context*, Ψ, is a finite function mapping each variable $x \in \text{dom}(\Psi)$ to an *expanded seTLM definition*, $x \hookrightarrow \mathtt{setlm}(\tau; e_{\text{parse}})$, where τ is the seTLM's type annotation, and e_{parse} is its parse function. We write $\Psi, x \hookrightarrow \mathtt{setlm}(\tau; e_{\text{parse}})$ when $x \notin \text{dom}(\Psi)$ for the extension of Ψ that maps x to $x \hookrightarrow \mathtt{setlm}(\tau; e_{\text{parse}})$. We write $\Delta \vdash \Psi\ \mathsf{seTLMs}$ when all the type annotations in Ψ are well-formed assuming Δ, and the parse functions in Ψ are closed and of the appropriate type.

Definition B.18 (seTLM Definition Context Formation)**.** $\Delta \vdash \Psi\ \mathsf{seTLMs}$ *iff for each* $x \hookrightarrow \mathtt{setlm}(\tau; e_{parse}) \in \Psi$, *we have* $\Delta \vdash \tau\ \mathsf{type}$ *and* $\emptyset\, \emptyset \vdash e_{parse} : \mathtt{parr}(\mathtt{Body}; \mathtt{ParseResultE})$.

Definition B.19 (seTLM Context Formation)**.** $\Delta \vdash \langle \mathcal{A}; \Psi \rangle\ \mathsf{seTLMctx}$ *iff* $\Delta \vdash \Psi\ \mathsf{seTLMs}$ *and for each* $\hat{a} \rightsquigarrow x \in \mathcal{A}$ *we have* $x \in dom(\Psi)$.

We define $\hat{\Psi}, \hat{a} \rightsquigarrow x \hookrightarrow \mathtt{setlm}(\tau; e_{\text{parse}})$, when $\hat{\Psi} = \langle \mathcal{A}; \Phi \rangle$, as an abbreviation of

$$\langle \mathcal{A} \uplus \hat{a} \rightsquigarrow x; \Psi, x \hookrightarrow \mathtt{setlm}(\tau; e_{\text{parse}}) \rangle$$

spTLM Contexts

spTLM contexts, $\hat{\Phi}$, are of the form $\langle \mathcal{A}; \Phi \rangle$, where $\mathcal{A}$ is a TLM identifier expansion context, defined above, and Φ is a *spTLM definition context*.

An *spTLM definition context*, Φ, is a finite function mapping each variable $x \in \text{dom}(\Phi)$ to an *expanded seTLM definition*, $a \hookrightarrow \texttt{sptlm}(\tau; e_{\text{parse}})$, where τ is the spTLM's type annotation, and e_{parse} is its parse function. We write $\Phi, a \hookrightarrow \texttt{sptlm}(\tau; e_{\text{parse}})$ when $a \notin \text{dom}(\Phi)$ for the extension of Φ that maps x to $a \hookrightarrow \texttt{sptlm}(\tau; e_{\text{parse}})$. We write $\Delta \vdash \Phi$ spTLMs when all the type annotations in Φ are well-formed assuming Δ, and the parse functions in Φ are closed and of the appropriate type.

Definition B.20 (spTLM Definition Context Formation). $\Delta \vdash \Phi$ spTLMs *iff for each* $a \hookrightarrow \texttt{sptlm}(\tau; e_{parse}) \in \Phi$, *we have* $\Delta \vdash \tau$ type *and* $\emptyset\,\emptyset \vdash e_{parse} : \texttt{parr}(\texttt{Body}; \texttt{ParseResultP})$.

Definition B.21 (spTLM Context Formation). $\Delta \vdash \langle \mathcal{A}; \Phi \rangle$ spTLMctx *iff* $\Delta \vdash \Phi$ spTLMs *and for each* $\hat{a} \rightsquigarrow x \in \mathcal{A}$ *we have* $x \in dom(\Phi)$.

We define $\hat{\Phi}, \hat{a} \rightsquigarrow x \hookrightarrow \texttt{sptlm}(\tau; e_{\text{parse}})$, when $\hat{\Phi} = \langle \mathcal{A}; \Phi \rangle$, as an abbreviation of

$$\langle \mathcal{A} \uplus \hat{a} \rightsquigarrow x; \Phi, a \hookrightarrow \texttt{sptlm}(\tau; e_{\text{parse}}) \rangle$$

Typed Expression Expansion

$\boxed{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau}$ $\hat{e}$ has expansion e of type τ

$$\frac{}{\hat{\Delta}\;\hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau \vdash_{\hat{\Psi};\hat{\Phi}} \hat{x} \rightsquigarrow x : \tau} \tag{B.6a}$$

$$\frac{\hat{\Delta} \vdash \hat{\tau} \rightsquigarrow \tau \text{ type} \qquad \hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau}{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} : \hat{\tau} \rightsquigarrow e : \tau} \tag{B.6b}$$

$$\frac{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_1 \rightsquigarrow e_1 : \tau_1 \qquad \hat{\Delta}\;\hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau_1 \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_2 \rightsquigarrow e_2 : \tau_2}{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \texttt{let}\;\texttt{val}\;\hat{x} = \hat{e}_1\;\texttt{in}\;\hat{e}_2 \rightsquigarrow \texttt{ap(lam\{}\tau_1\texttt{\}}(x.e_2)\texttt{;}e_1\texttt{)} : \tau_2} \tag{B.6c}$$

$$\frac{\hat{\Delta} \vdash \hat{\tau} \rightsquigarrow \tau \text{ type} \qquad \hat{\Delta}\;\hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau'}{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \lambda \hat{x}{:}\hat{\tau}.\hat{e} \rightsquigarrow \texttt{lam\{}\tau\texttt{\}}(x.e) : \texttt{parr}(\tau;\tau')} \tag{B.6d}$$

$$\frac{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_1 \rightsquigarrow e_1 : \texttt{parr}(\tau;\tau') \qquad \hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_2 \rightsquigarrow e_2 : \tau}{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_1(\hat{e}_2) \rightsquigarrow \texttt{ap}(e_1;e_2) : \tau'} \tag{B.6e}$$

$$\frac{\hat{\Delta}, \hat{t} \rightsquigarrow t \text{ type}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau}{\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \Lambda\hat{t}.\hat{e} \rightsquigarrow \texttt{tlam}(t.e) : \texttt{all}(t.\tau)} \tag{B.6f}$$

$$\frac{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \mathtt{all}(t.\tau) \qquad \hat{\Delta} \vdash \hat{\tau}' \rightsquigarrow \tau' \text{ type}}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}\mathtt{[}\hat{\tau}'\mathtt{]} \rightsquigarrow \mathtt{tap}\{\tau'\}\mathtt{(}e\mathtt{)} : [\tau'/t]\tau} \tag{B.6g}$$

$$\frac{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : [\mathtt{rec}(t.\tau)/t]\tau}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \mathtt{fold}(\hat{e}) \rightsquigarrow \mathtt{fold}(e) : \mathtt{rec}(t.\tau)} \tag{B.6h}$$

$$\frac{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \mathtt{rec}(t.\tau)}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \mathtt{unfold}(\hat{e}) \rightsquigarrow \mathtt{unfold}(e) : [\mathtt{rec}(t.\tau)/t]\tau} \tag{B.6i}$$

$$\frac{\{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_i \rightsquigarrow e_i : \tau_i\}_{i \in L}}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \langle \{i \hookrightarrow \hat{e}_i\}_{i \in L} \rangle \rightsquigarrow \mathtt{tpl}(\{i \hookrightarrow e_i\}_{i \in L}) : \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L})} \tag{B.6j}$$

$$\frac{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau)}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \cdot \ell \rightsquigarrow \mathtt{prj}[\ell](e) : \tau} \tag{B.6k}$$

$$\frac{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau'}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \mathtt{inj}[\ell](\hat{e}) \rightsquigarrow \mathtt{inj}[\ell](e) : \mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau')} \tag{B.6l}$$

$$\frac{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}) \qquad \{\hat{\Delta}\ \hat{\Gamma}, \hat{x}_i \rightsquigarrow x_i : \tau_i \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_i \rightsquigarrow e_i : \tau\}_{i \in L}}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \mathtt{case}\ \hat{e}\ \{i \hookrightarrow \hat{x}_i.\hat{e}_i\}_{i \in L} \rightsquigarrow \mathtt{case}(e; \{i \hookrightarrow x_i.e_i\}_{i \in L}) : \tau} \tag{B.6m}$$

$$\frac{\begin{array}{c}\hat{\Delta} \vdash \hat{\tau} \rightsquigarrow \tau \text{ type} \\ \emptyset\, \emptyset \vdash e_{\text{parse}} : \mathtt{parr(Body;ParseResultE)} \qquad \hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_{\text{dep}} \rightsquigarrow e_{\text{dep}} : \tau_{\text{dep}} \\ \hat{\Gamma} = \langle \mathcal{G}; \Gamma \rangle \qquad \hat{\Delta}\ \langle \mathcal{G}; \Gamma, x : \tau_{\text{dep}} \rangle \vdash_{\hat{\Psi}, \hat{a} \rightsquigarrow x \hookrightarrow \mathtt{setlm}(\tau; e_{\text{parse}}); \hat{\Phi}} \hat{e} \rightsquigarrow e : \tau' \\ e_{\text{defn}} = \mathtt{ap(lam}\{\tau_{\text{dep}}\}\mathtt{(}x.e\mathtt{)};e_{\text{dep}}\mathtt{)}\end{array}}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \mathtt{notation}\ \hat{a}\ \mathtt{at}\ \hat{\tau}\ \{\ \mathtt{expr\ parser}\ e_{\text{parse}}; \mathtt{expansions\ require}\ \hat{e}\ \}\ \mathtt{in}\ \hat{e} \rightsquigarrow e_{\text{defn}} : \tau'} \tag{B.6n}$$

$$\frac{\begin{array}{c}\hat{\Psi} = \hat{\Psi}', \hat{a} \rightsquigarrow x \hookrightarrow \mathtt{setlm}(\tau; e_{\text{parse}}) \qquad \hat{\Gamma} = \langle \mathcal{G}; \Gamma, x : \tau_{\text{dep}} \rangle \\ b \downarrow_{\mathsf{Body}} e_{\text{body}} \qquad e_{\text{parse}}(e_{\text{body}}) \Downarrow \mathtt{inj[SuccessE]}(e_{\text{proto}}) \qquad e_{\text{proto}} \uparrow_{\mathsf{PrExpr}} \grave{e} \\ \mathsf{seg}(\grave{e}) \text{ segments } b \qquad \emptyset\, \emptyset \vdash^{\hat{\Delta}; \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b} \grave{e} \rightsquigarrow e : \mathtt{parr}(\tau_{\text{dep}}; \tau)\end{array}}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{a}\ \mathtt{`(}b\mathtt{)`} \rightsquigarrow \mathtt{ap}(e; x) : \tau} \tag{B.6o}$$

$$\frac{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau \qquad \{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{r}_i \rightsquigarrow r_i : \tau \Rightarrow \tau'\}_{1 \le i \le n}}{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \mathtt{match}\ \hat{e}\ \{\hat{r}_i\}_{1 \le i \le n} \rightsquigarrow \mathtt{match}(e; \{r_i\}_{1 \le i \le n}) : \tau'} \tag{B.6p}$$

$$\frac{\hat{\Delta} \vdash \hat{\tau} \leadsto \tau \text{ type} \qquad \emptyset\,\emptyset \vdash e_{\text{parse}} : \texttt{parr(Body;ParseResultP)} \qquad \hat{\Delta}\,\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi},\hat{a}\leadsto x \hookrightarrow \texttt{sptlm}(\tau;e'_{\text{parse}})} \hat{e} \leadsto e : \tau'}{\hat{\Delta}\,\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \texttt{notation}\ \hat{a}\ \texttt{at}\ \hat{\tau}\ \{\ \texttt{pat parser}\ e_{\text{parse}}\ \}\ \texttt{in}\ \hat{e} \leadsto e : \tau'} \tag{B.6q}$$

$\boxed{\hat{\Delta}\,\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{r} \leadsto r : \tau \Mapsto \tau'}$ $\hat{r}$ has expansion r taking values of type τ to values of type τ'

$$\frac{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \leadsto p : \tau \dashv\!\mid \langle \mathcal{G}';\Gamma' \rangle \qquad \hat{\Delta}\ \langle \mathcal{G} \uplus \mathcal{G}';\Gamma \cup \Gamma' \rangle \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \leadsto e : \tau'}{\hat{\Delta}\ \langle \mathcal{G};\Gamma \rangle \vdash_{\hat{\Psi};\hat{\Phi}} \texttt{urule}(\hat{p}.\hat{e}) \leadsto \texttt{rule}(p.e) : \tau \Mapsto \tau'} \tag{B.7}$$

Typed Pattern Expansion

$\boxed{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \leadsto p : \tau \dashv\!\mid \hat{\Gamma}}$ $\hat{p}$ has expansion p matching against τ generating hypotheses $\hat{\Gamma}$

$$\frac{}{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{x} \leadsto x : \tau \dashv\!\mid \langle \hat{x} \leadsto x; x : \tau \rangle} \tag{B.8a}$$

$$\frac{}{\hat{\Delta} \vdash_{\hat{\Phi}} \_ \leadsto \texttt{wildp} : \tau \dashv\!\mid \langle \emptyset;\emptyset \rangle} \tag{B.8b}$$

$$\frac{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \leadsto p : [\texttt{rec}(t.\tau)/t]\tau \dashv\!\mid \hat{\Gamma}}{\hat{\Delta} \vdash_{\hat{\Phi}} \texttt{fold}(\hat{p}) \leadsto \texttt{foldp}(p) : \texttt{rec}(t.\tau) \dashv\!\mid \hat{\Gamma}} \tag{B.8c}$$

$$\frac{\tau = \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i\in L}) \qquad \{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p}_i \leadsto p_i : \tau_i \dashv\!\mid \hat{\Gamma}_i\}_{i\in L}}{\hat{\Delta} \vdash_{\hat{\Phi}} \langle \{i \hookrightarrow \hat{p}_i\}_{i\in L} \rangle \leadsto \texttt{tplp}(\{i \hookrightarrow p_i\}_{i\in L}) : \tau \dashv\!\mid \uplus_{i\in L} \hat{\Gamma}_i} \tag{B.8d}$$

$$\frac{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \leadsto p : \tau \dashv\!\mid \hat{\Gamma}}{\hat{\Delta} \vdash_{\hat{\Phi}} \texttt{inj}[\ell](\hat{p}) \leadsto \texttt{injp}[\ell](p) : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L}; \ell \hookrightarrow \tau) \dashv\!\mid \hat{\Gamma}} \tag{B.8e}$$

$$\frac{\hat{\Phi} = \hat{\Phi}', \hat{a} \leadsto \_ \hookrightarrow \texttt{sptlm}(\tau;\, e_{\text{parse}}) \qquad b \downarrow_{\text{Body}} e_{\text{body}} \qquad e_{\text{parse}}(e_{\text{body}}) \Downarrow \texttt{inj}[\texttt{SuccessP}](e_{\text{proto}}) \qquad e_{\text{proto}} \uparrow_{\text{PrPat}} \grave{p} \qquad \text{seg}(\grave{p}) \text{ segments } b \qquad \grave{p} \leadsto p : \tau \dashv\!\mid^{\hat{\Delta};\hat{\Phi};b} \hat{\Gamma}}{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{a}\ \text{`}(b)\text{`} \leadsto p : \tau \dashv\!\mid \hat{\Gamma}} \tag{B.8f}$$

In Rule (B.8d), $\hat{\Gamma}_i$ is shorthand for $\langle \mathcal{G}_i;\Gamma_i \rangle$ and $\uplus_{i\in L}\hat{\Gamma}_i$ is shorthand for

$$\langle \uplus_{i\in L}\mathcal{G}_i; \cup_{i\in L}\Gamma_i \rangle$$

B.4 Proto-Expansion Validation

B.4.1 Syntax of Proto-Expansions

Sort		Operational Form	Stylized Form	Description
PrTyp	$\grave{\tau}$ ::=	t	t	variable
		$\texttt{prparr}(\grave{\tau};\grave{\tau})$	$\grave{\tau} \rightharpoonup \grave{\tau}$	partial function
		$\texttt{prall}(t.\grave{\tau})$	$\forall t.\grave{\tau}$	polymorphic
		$\texttt{prrec}(t.\grave{\tau})$	$\mu t.\grave{\tau}$	recursive
		$\texttt{prprod}(\{i \hookrightarrow \grave{\tau}_i\}_{i\in L})$	$\langle\{i \hookrightarrow \grave{\tau}_i\}_{i\in L}\rangle$	labeled product
		$\texttt{prsum}(\{i \hookrightarrow \grave{\tau}_i\}_{i\in L})$	$[\{i \hookrightarrow \grave{\tau}_i\}_{i\in L}]$	labeled sum
		$\texttt{splicedt}[m;n]$	$\texttt{splicedt}[m;n]$	spliced type ref.
PrExp	$\grave{e}$ ::=	x	x	variable
		$\texttt{prasc}\{\grave{\tau}\}(\grave{e})$	$\grave{e} : \grave{\tau}$	ascription
		$\texttt{prletval}(\grave{e};x.\grave{e})$	$\texttt{let}\ \texttt{val}\ x = \grave{e}\ \texttt{in}\ \grave{e}$	value binding
		$\texttt{prlam}\{\grave{\tau}\}(x.\grave{e})$	$\lambda x{:}\grave{\tau}.\grave{e}$	abstraction
		$\texttt{prap}(\grave{e};\grave{e})$	$\grave{e}(\grave{e})$	application
		$\texttt{prtlam}(t.\grave{e})$	$\Lambda t.\grave{e}$	type abstraction
		$\texttt{prtap}\{\grave{\tau}\}(\grave{e})$	$\grave{e}[\grave{\tau}]$	type application
		$\texttt{prfold}(\grave{e})$	$\texttt{fold}(\grave{e})$	fold
		$\texttt{prunfold}(\grave{e})$	$\texttt{unfold}(\grave{e})$	unfold
		$\texttt{prtpl}(\{i \hookrightarrow \grave{e}_i\}_{i\in L})$	$\langle\{i \hookrightarrow \grave{e}_i\}_{i\in L}\rangle$	labeled tuple
		$\texttt{prprj}[\ell](\grave{e})$	$\grave{e} \cdot \ell$	projection
		$\texttt{prinj}[\ell](\grave{e})$	$\texttt{inj}[\ell](\grave{e})$	injection
		$\texttt{prcase}(\grave{e};\{i \hookrightarrow x_i.\grave{e}_i\}_{i\in L})$	$\texttt{case}\ \grave{e}\ \{i \hookrightarrow x_i.\grave{e}_i\}_{i\in L}$	case analysis
		$\texttt{splicede}[m;n;\grave{\tau}]$	$\texttt{splicede}[m;n;\grave{\tau}]$	spliced expr. ref.
		$\texttt{prmatch}(\grave{e};\{\grave{r}_i\}_{1\le i\le n})$	$\texttt{match}\ \grave{e}\ \{\grave{r}_i\}_{1\le i\le n}$	match
PrRule	$\grave{r}$ ::=	$\texttt{prrule}(p.\grave{e})$	$p \Rightarrow \grave{e}$	rule
PrPat	$\grave{p}$ ::=	$\texttt{prwildp}$	_	wildcard pattern
		$\texttt{prfoldp}(\grave{p})$	$\texttt{fold}(\grave{p})$	fold pattern
		$\texttt{prtplp}[L](\{i \hookrightarrow \grave{p}_i\}_{i\in L})$	$\langle\{i \hookrightarrow \grave{p}_i\}_{i\in L}\rangle$	labeled tuple pattern
		$\texttt{prinjp}[\ell](\grave{p})$	$\texttt{inj}[\ell](\grave{p})$	injection pattern
		$\texttt{splicedp}[m;n;\grave{\tau}]$	$\texttt{splicedp}[m;n;\grave{\tau}]$	spliced pattern ref.

Common Proto-Expansion Terms

Each core language term, except variable patterns, maps onto a proto-expansion term. We refer to these as the *common proto-expansion terms*. In particular:

- Each type, τ, maps onto a proto-type, $\mathcal{P}(\tau)$, as follows:

$$
\begin{aligned}
\mathcal{P}(t) &= t\\
\mathcal{P}(\texttt{parr}(\tau_1;\tau_2)) &= \texttt{prparr}(\mathcal{P}(\tau_1);\mathcal{P}(\tau_2))\\
\mathcal{P}(\texttt{all}(t.\tau)) &= \texttt{prall}(t.\mathcal{P}(\tau))\\
\mathcal{P}(\texttt{rec}(t.\tau)) &= \texttt{prrec}(t.\mathcal{P}(\tau))\\
\mathcal{P}(\texttt{prod}(\{i \hookrightarrow \tau_i\}_{i\in L})) &= \texttt{prprod}(\{i \hookrightarrow \mathcal{P}(\tau_i)\}_{i\in L})\\
\mathcal{P}(\texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L})) &= \texttt{prsum}(\{i \hookrightarrow \mathcal{P}(\tau_i)\}_{i\in L})
\end{aligned}
$$

- Each core language expression, e, maps onto a proto-expression, $\mathcal{P}(e)$, as follows:

$$
\begin{aligned}
\mathcal{P}(x) &= x\\
\mathcal{P}(\texttt{lam}\{\tau\}(x.e)) &= \texttt{prlam}\{\mathcal{P}(\tau)\}(x.\mathcal{P}(e))\\
\mathcal{P}(\texttt{ap}(e_1;e_2)) &= \texttt{prap}(\mathcal{P}(e_1);\mathcal{P}(e_2))\\
\mathcal{P}(\texttt{tlam}(t.e)) &= \texttt{prtlam}(t.\mathcal{P}(e))\\
\mathcal{P}(\texttt{tap}\{\tau\}(e)) &= \texttt{prtap}\{\mathcal{P}(\tau)\}(\mathcal{P}(e))\\
\mathcal{P}(\texttt{fold}(e)) &= \texttt{prfold}(\mathcal{P}(e))\\
\mathcal{P}(\texttt{unfold}(e)) &= \texttt{prunfold}(\mathcal{P}(e))\\
\mathcal{P}(\texttt{tpl}(\{i \hookrightarrow e_i\}_{i\in L})) &= \texttt{prtpl}(\{i \hookrightarrow \mathcal{P}(e_i)\}_{i\in L})\\
\mathcal{P}(\texttt{inj}[\ell](e)) &= \texttt{prinj}[\ell](\mathcal{P}(e))\\
\mathcal{P}(\texttt{match}(e;\{r_i\}_{1\leq i\leq n})) &= \texttt{prmatch}(\mathcal{P}(e);\{\mathcal{P}(r_i)\}_{1\leq i\leq n})
\end{aligned}
$$

- Each core language rule, r, maps onto the proto-rule, $\mathcal{P}(r)$, as follows:

$$\mathcal{P}(\texttt{rule}(p.e)) = \texttt{prrule}(p.\mathcal{P}(e))$$

  Notice that proto-rules bind expanded patterns, not proto-patterns. This is because proto-rules appear in proto-expressions, which are generated by seTLMs. It would not be sensible for an seTLM to splice a pattern out of a literal body.

- Each core language pattern, p, except for the variable patterns, maps onto a proto-pattern, $\mathcal{P}(p)$, as follows:

$$
\begin{aligned}
\mathcal{P}(\texttt{wildp}) &= \texttt{prwildp}\\
\mathcal{P}(\texttt{foldp}(p)) &= \texttt{prfoldp}(\mathcal{P}(p))\\
\mathcal{P}(\texttt{tplp}(\{i \hookrightarrow p_i\}_{i\in L})) &= \texttt{prtplp}[L](\{i \hookrightarrow \mathcal{P}(p_i)\}_{i\in L})\\
\mathcal{P}(\texttt{injp}[\ell](p)) &= \texttt{prinjp}[\ell](\mathcal{P}(p))
\end{aligned}
$$

Proto-Expression Encoding and Decoding

The type abbreviated PrExpr classifies encodings of *proto-expressions*. The mapping from proto-expressions to values of type PrExpr is defined by the *proto-expression encoding judgement*, $\grave{e} \downarrow_{\mathsf{PrExpr}} e$. An inverse mapping is defined by the *proto-expression decoding judgement*, $e \uparrow_{\mathsf{PrExpr}} \grave{e}$.

Judgement Form	Description
$\grave{e} \downarrow_{\mathsf{PrExpr}} e$	$\grave{e}$ has encoding e
$e \uparrow_{\mathsf{PrExpr}} \grave{e}$	e has decoding $\grave{e}$

Rather than picking a particular definition of `PrExpr` and defining the judgements above inductively against it, we only state the following condition, which establishes an isomorphism between values of type `PrExpr` and proto-expressions.

Condition B.22 (Proto-Expression Isomorphism)**.**

1. *For every $\grave{e}$, we have $\grave{e} \downarrow_{\mathsf{PrExpr}} e_{proto}$ for some e_{proto} such that $\vdash e_{proto} : \texttt{PrExpr}$ and $e_{proto}\ \mathsf{val}$.*
2. *If $\vdash e_{proto} : \texttt{PrExpr}$ and $e_{proto}\ \mathsf{val}$ then $e_{proto} \uparrow_{\mathsf{PrExpr}} \grave{e}$ for some $\grave{e}$.*
3. *If $\grave{e} \downarrow_{\mathsf{PrExpr}} e_{proto}$ then $e_{proto} \uparrow_{\mathsf{PrExpr}} \grave{e}$.*
4. *If $\vdash e_{proto} : \texttt{PrExpr}$ and $e_{proto}\ \mathsf{val}$ and $e_{proto} \uparrow_{\mathsf{PrExpr}} \grave{e}$ then $\grave{e} \downarrow_{\mathsf{PrExpr}} e_{proto}$.*
5. *If $\grave{e} \downarrow_{\mathsf{PrExpr}} e_{proto}$ and $\grave{e} \downarrow_{\mathsf{PrExpr}} e'_{proto}$ then $e_{proto} = e'_{proto}$.*
6. *If $\vdash e_{proto} : \texttt{PrExpr}$ and $e_{proto}\ \mathsf{val}$ and $e_{proto} \uparrow_{\mathsf{PrExpr}} \grave{e}$ and $e_{proto} \uparrow_{\mathsf{PrExpr}} \grave{e}'$ then $\grave{e} = \grave{e}'$.*

Proto-Pattern Encoding and Decoding

The type abbreviated `PrPat` classifies encodings of *proto-patterns*. The mapping from proto-patterns to values of type `PrPat` is defined by the *proto-pattern encoding judgement*, $\grave{p} \downarrow_{\mathsf{PrPat}} p$. An inverse mapping is defined by the *proto-expression decoding judgement*, $p \uparrow_{\mathsf{PrPat}} \grave{p}$.

Judgement Form	Description
$\grave{p} \downarrow_{\mathsf{PrPat}} p$	$\grave{p}$ has encoding p
$p \uparrow_{\mathsf{PrPat}} \grave{p}$	p has decoding $\grave{p}$

Again, rather than picking a particular definition of `PrPat` and defining the judgements above inductively against it, we only state the following condition, which establishes an isomorphism between values of type `PrPat` and proto-patterns.

Condition B.23 (Proto-Pattern Isomorphism)**.**

1. *For every $\grave{p}$, we have $\grave{p} \downarrow_{\mathsf{PrPat}} e_{proto}$ for some e_{proto} such that $\vdash e_{proto} : \texttt{PrPat}$ and $e_{proto}\ \mathsf{val}$.*
2. *If $\vdash e_{proto} : \texttt{PrPat}$ and $e_{proto}\ \mathsf{val}$ then $e_{proto} \uparrow_{\mathsf{PrPat}} \grave{p}$ for some $\grave{p}$.*
3. *If $\grave{p} \downarrow_{\mathsf{PrPat}} e_{proto}$ then $e_{proto} \uparrow_{\mathsf{PrPat}} \grave{p}$.*
4. *If $\vdash e_{proto} : \texttt{PrPat}$ and $e_{proto}\ \mathsf{val}$ and $e_{proto} \uparrow_{\mathsf{PrPat}} \grave{p}$ then $\grave{p} \downarrow_{\mathsf{PrPat}} e_{proto}$.*
5. *If $\grave{p} \downarrow_{\mathsf{PrPat}} e_{proto}$ and $\grave{p} \downarrow_{\mathsf{PrPat}} e'_{proto}$ then $e_{proto} = e'_{proto}$.*

6. *If* $\vdash e_{proto} : \mathtt{PrPat}$ *and* e_{proto} val *and* $e_{proto} \uparrow_{\mathsf{PrPat}} \grave{p}$ *and* $e_{proto} \uparrow_{\mathsf{PrPat}} \grave{p}'$ *then* $\grave{p} = \grave{p}'$.

Segmentations

The *segmentation*, ψ, of a proto-type, $\mathsf{seg}(\grave{\tau})$ or proto-expression, $\mathsf{seg}(\grave{e})$, is the finite set of references to spliced types and expressions that it mentions.

$$
\begin{array}{lcl}
\mathsf{seg}(t) & = & \emptyset \\
\mathsf{seg}(\mathtt{prparr}(\grave{\tau}_1;\grave{\tau}_2)) & = & \mathsf{seg}(\grave{\tau}_1) \cup \mathsf{seg}(\grave{\tau}_2) \\
\mathsf{seg}(\mathtt{prall}(t.\grave{\tau})) & = & \mathsf{seg}(\grave{\tau}) \\
\mathsf{seg}(\mathtt{prrec}(t.\grave{\tau})) & = & \mathsf{seg}(\grave{\tau}) \\
\mathsf{seg}(\mathtt{prprod}(\{i \hookrightarrow \grave{\tau}_i\}_{i\in L})) & = & \bigcup_{i\in L} \mathsf{seg}(\grave{\tau}_i) \\
\mathsf{seg}(\mathtt{prsum}(\{i \hookrightarrow \grave{\tau}_i\}_{i\in L})) & = & \bigcup_{i\in L} \mathsf{seg}(\grave{\tau}_i) \\
\mathsf{seg}(\mathtt{splicedt}[m;n]) & = & \{\mathtt{splicedt}[m;n]\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{seg}(x) & = & \emptyset \\
\mathsf{seg}(\mathtt{prasc}\{\grave{\tau}\}(\grave{e})) & = & \mathsf{seg}(\grave{\tau}) \cup \mathsf{seg}(\grave{e}) \\
\mathsf{seg}(\mathtt{prletval}(\grave{e}_1;x.\grave{e}_2)) & = & \mathsf{seg}(\grave{e}_1) \cup \mathsf{seg}(\grave{e}_2) \\
\mathsf{seg}(\mathtt{prlam}\{\grave{\tau}\}(x.\grave{e})) & = & \mathsf{seg}(\grave{\tau}) \cup \mathsf{seg}(\grave{e}) \\
\mathsf{seg}(\mathtt{prap}(\grave{e}_1;\grave{e}_2)) & = & \mathsf{seg}(\grave{e}_1) \cup \mathsf{seg}(\grave{e}_2) \\
\mathsf{seg}(\mathtt{prtlam}(t.\grave{e})) & = & \mathsf{seg}(\grave{e}) \\
\mathsf{seg}(\mathtt{prtap}\{\grave{\tau}\}(\grave{e})) & = & \mathsf{seg}(\grave{e}) \cup \mathsf{seg}(\grave{\tau}) \\
\mathsf{seg}(\mathtt{prfold}(\grave{e})) & = & \mathsf{seg}(\grave{e}) \\
\mathsf{seg}(\mathtt{prunfold}(\grave{e})) & = & \mathsf{seg}(\grave{e}) \\
\mathsf{seg}(\mathtt{prtpl}(\{i \hookrightarrow x_i.\grave{e}_i\}_{i\in L})) & = & \bigcup_{i\in L} \mathsf{seg}(\grave{e}_i) \\
\mathsf{seg}(\mathtt{prprj}[\ell](\grave{e})) & = & \mathsf{seg}(\grave{e}) \\
\mathsf{seg}(\mathtt{prinj}[\ell](\grave{e})) & = & \mathsf{seg}(\grave{e}) \\
\mathsf{seg}(\mathtt{prcase}(\grave{e};\{i \hookrightarrow x_i.\grave{e}_i\}_{i\in L})) & = & \mathsf{seg}(\grave{e}) \cup \bigcup_{i\in L} \mathsf{seg}(\grave{e}_i) \\
\mathsf{seg}(\mathtt{splicede}[m;n;\grave{\tau}]) & = & \{\mathtt{splicede}[m;n;\grave{\tau}]\} \cup \mathsf{seg}(\grave{\tau}) \\
\mathsf{seg}(\mathtt{prmatch}(\grave{e};\{\grave{r}_i\}_{1\le i\le n})) & = & \mathsf{seg}(\grave{e}) \cup \bigcup_{1\le i\le n} \mathsf{seg}(\grave{r}_i) \\
\mathsf{seg}(\mathtt{prrule}(p.\grave{e})) & = & \mathsf{seg}(\grave{e})
\end{array}
$$

The splice summary of a proto-pattern, $\mathsf{seg}(\grave{p})$, is the finite set of references to spliced types and patterns that it mentions.

$$
\begin{array}{lcl}
\mathsf{seg}(\mathtt{prwildp}) & = & \emptyset \\
\mathsf{seg}(\mathtt{prfoldp}(\grave{p})) & = & \mathsf{seg}(\grave{p}) \\
\mathsf{seg}(\mathtt{prtplp}[L](\{i \hookrightarrow \grave{p}_i\}_{i\in L})) & = & \bigcup_{i\in L} \mathsf{seg}(\grave{p}_i) \\
\mathsf{seg}(\mathtt{prinjp}[\ell](\grave{p})) & = & \mathsf{seg}(\grave{p}) \\
\mathsf{seg}(\mathtt{splicedp}[m;n;\grave{\tau}]) & = & \{\mathtt{splicedp}[m;n;\grave{\tau}]\} \cup \mathsf{seg}(\grave{\tau})
\end{array}
$$

The predicate ψ segments b defined below checks that each segment in ψ, has positive extent and is within bounds of b, and that the segments in ψ do not overlap or sit immediately adjacent to one another, and that spliced segments that are exactly overlapping have equal segment types.

Definition B.24 (Segmentation Validity)**.** ψ segments b *iff*

1. *For each* $\texttt{splicedt}[m;n] \in \psi$*, all of the following hold:*
   (a) $0 \leq m \leq n < \|b\|$
   (b) *For each* $\texttt{splicedt}[m';n'] \in \psi$*, either*
      i. $m = m'$ *and* $n = n'$*; or*
      ii. $n' < m - 1$*; or*
      iii. $m' > n + 1$
   (c) *For each* $\texttt{splicede}[m';n';\check{\tau}] \in \psi$*, either*
      i. $n' < m - 1$*; or*
      ii. $m' > n + 1$
   (d) *For each* $\texttt{splicedp}[m';n';\check{\tau}] \in \psi$*, either*
      i. $n' < m - 1$*; or*
      ii. $m' > n + 1$
2. *For each* $\texttt{splicede}[m;n;\check{\tau}] \in \psi$*, all of the following hold:*
   (a) $0 \leq m \leq n < \|b\|$
   (b) *For each* $\texttt{splicedt}[m';n'] \in \psi$*, either*
      i. $n' < m - 1$*; or*
      ii. $m' > n + 1$
   (c) *For each* $\texttt{splicede}[m';n';\check{\tau}'] \in \psi$*, either*
      i. $m = m'$ *and* $n = n'$ *and* $\check{\tau} = \check{\tau}'$*; or*
      ii. $n' < m - 1$*; or*
      iii. $m' > n + 1$
3. *For each* $\texttt{splicedp}[m;n;\check{\tau}] \in \psi$*, all of the following hold:*
   (a) $0 \leq m \leq n < \|b\|$
   (b) *For each* $\texttt{splicedt}[m';n'] \in \psi$*, either*
      i. $n' < m - 1$*; or*
      ii. $m' > n + 1$
   (c) *For each* $\texttt{splicede}[m';n';\check{\tau}'] \in \psi$*, either*
      i. $n' < m - 1$*; or*
      ii. $m' > n + 1$
   (d) *For each* $\texttt{splicedp}[m';n';\check{\tau}'] \in \psi$*, either*

i. $m = m'$ *and* $n = n'$ *and* $\grave{\tau} = \grave{\tau}'$; *or*

ii. $n' < m - 1$; *or*

iii. $m' > n + 1$

B.4.2 Proto-Type Validation

Type splicing scenes, $\mathbb{T}$, are of the form $\hat{\Delta}; b$.

$\boxed{\Delta \vdash^{\mathbb{T}} \grave{\tau} \rightsquigarrow \tau \ \mathsf{type}}$ $\grave{\tau}$ has well-formed expansion τ

$$\frac{}{\Delta, t \ \mathsf{type} \vdash^{\mathbb{T}} t \rightsquigarrow t \ \mathsf{type}} \tag{B.9a}$$

$$\frac{\Delta \vdash^{\mathbb{T}} \grave{\tau}_1 \rightsquigarrow \tau_1 \ \mathsf{type} \qquad \Delta \vdash^{\mathbb{T}} \grave{\tau}_2 \rightsquigarrow \tau_2 \ \mathsf{type}}{\Delta \vdash^{\mathbb{T}} \mathtt{prparr}(\grave{\tau}_1; \grave{\tau}_2) \rightsquigarrow \mathtt{parr}(\tau_1; \tau_2) \ \mathsf{type}} \tag{B.9b}$$

$$\frac{\Delta, t \ \mathsf{type} \vdash^{\mathbb{T}} \grave{\tau} \rightsquigarrow \tau \ \mathsf{type}}{\Delta \vdash^{\mathbb{T}} \mathtt{prall}(t.\grave{\tau}) \rightsquigarrow \mathtt{all}(t.\tau) \ \mathsf{type}} \tag{B.9c}$$

$$\frac{\Delta, t \ \mathsf{type} \vdash^{\mathbb{T}} \grave{\tau} \rightsquigarrow \tau \ \mathsf{type}}{\Delta \vdash^{\mathbb{T}} \mathtt{prrec}(t.\grave{\tau}) \rightsquigarrow \mathtt{rec}(t.\tau) \ \mathsf{type}} \tag{B.9d}$$

$$\frac{\{\Delta \vdash^{\mathbb{T}} \grave{\tau}_i \rightsquigarrow \tau_i \ \mathsf{type}\}_{i \in L}}{\Delta \vdash^{\mathbb{T}} \mathtt{prprod}(\{i \hookrightarrow \grave{\tau}_i\}_{i \in L}) \rightsquigarrow \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}) \ \mathsf{type}} \tag{B.9e}$$

$$\frac{\{\Delta \vdash^{\mathbb{T}} \grave{\tau}_i \rightsquigarrow \tau_i \ \mathsf{type}\}_{i \in L}}{\Delta \vdash^{\mathbb{T}} \mathtt{prsum}(\{i \hookrightarrow \grave{\tau}_i\}_{i \in L}) \rightsquigarrow \mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}) \ \mathsf{type}} \tag{B.9f}$$

$$\frac{\mathsf{parseUTyp}(\mathsf{subseq}(b; m; n)) = \hat{\tau} \qquad \langle \mathcal{D}; \Delta_{\mathrm{app}} \rangle \vdash \hat{\tau} \rightsquigarrow \tau \ \mathsf{type} \qquad \Delta \cap \Delta_{\mathrm{app}} = \emptyset}{\Delta \vdash^{\langle \mathcal{D}; \Delta_{\mathrm{app}} \rangle; b} \mathtt{splicedt}[m; n] \rightsquigarrow \tau \ \mathsf{type}} \tag{B.9g}$$

B.4.3 Proto-Expression Validation

Expression splicing scenes, $\mathbb{E}$, are of the form $\hat{\Delta}; \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b$. We write $\mathsf{ts}(\mathbb{E})$ for the type splicing scene constructed by dropping unnecessary contexts from $\mathbb{E}$:

$$\mathsf{ts}(\hat{\Delta}; \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b) = \hat{\Delta}; b$$

$\boxed{\Delta \, \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau}$ $\grave{e}$ has expansion e of type τ

$$\frac{}{\Delta \, \Gamma, x : \tau \vdash^{\mathbb{E}} x \rightsquigarrow x : \tau} \tag{B.10a}$$

$$\frac{\Delta \vdash^{\mathsf{ts}(\mathbb{E})} \grave{\tau} \rightsquigarrow \tau \ \mathsf{type} \qquad \Delta \, \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau}{\Delta \, \Gamma \vdash^{\mathbb{E}} \mathtt{prasc}\{\grave{\tau}\}(\grave{e}) \rightsquigarrow e : \tau} \tag{B.10b}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e}_1 \rightsquigarrow e_1 : \tau_1 \qquad \Delta\ \Gamma, x : \tau_1 \vdash^{\grave{e}_2} e_2 \rightsquigarrow \tau_2 :}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prletval}(\grave{e}_1; x.\grave{e}_2) \rightsquigarrow \texttt{ap}(\texttt{lam}\{\tau_1\}(x.e_2); e_1) : \tau_2} \tag{B.10c}$$

$$\frac{\Delta \vdash^{\mathsf{ts}(\mathbb{E})} \grave{\tau} \rightsquigarrow \tau\ \mathsf{type} \qquad \Delta\ \Gamma, x : \tau \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau'}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prlam}\{\grave{\tau}\}(x.\grave{e}) \rightsquigarrow \texttt{lam}\{\tau\}(x.e) : \texttt{parr}(\tau; \tau')} \tag{B.10d}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e}_1 \rightsquigarrow e_1 : \texttt{parr}(\tau; \tau') \qquad \Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e}_2 \rightsquigarrow e_2 : \tau}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prap}(\grave{e}_1; \grave{e}_2) \rightsquigarrow \texttt{ap}(e_1; e_2) : \tau'} \tag{B.10e}$$

$$\frac{\Delta, t\ \mathsf{type}\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prtlam}(t.\grave{e}) \rightsquigarrow \texttt{tlam}(t.e) : \texttt{all}(t.\tau)} \tag{B.10f}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \texttt{all}(t.\tau) \qquad \Delta \vdash^{\mathsf{ts}(\mathbb{E})} \grave{\tau}' \rightsquigarrow \tau'\ \mathsf{type}}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prtap}\{\grave{\tau}'\}(\grave{e}) \rightsquigarrow \texttt{tap}\{\tau'\}(e) : [\tau'/t]\tau} \tag{B.10g}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : [\texttt{rec}(t.\tau)/t]\tau}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prfold}(\grave{e}) \rightsquigarrow \texttt{fold}(e) : \texttt{rec}(t.\tau)} \tag{B.10h}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \texttt{rec}(t.\tau)}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prunfold}(\grave{e}) \rightsquigarrow \texttt{unfold}(e) : [\texttt{rec}(t.\tau)/t]\tau} \tag{B.10i}$$

$$\frac{\begin{array}{c}\tau = \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}) \\ \{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e}_i \rightsquigarrow e_i : \tau_i\}_{i \in L}\end{array}}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prtpl}(\{i \hookrightarrow \grave{e}_i\}_{i \in L}) \rightsquigarrow \texttt{tpl}(\{i \hookrightarrow e_i\}_{i \in L}) : \tau} \tag{B.10j}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau)}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prprj}[\ell](\grave{e}) \rightsquigarrow \texttt{prj}[\ell](e) : \tau} \tag{B.10k}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau'}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prinj}[\ell](\grave{e}) \rightsquigarrow \texttt{inj}[\ell](e) : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau')} \tag{B.10l}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}) \qquad \{\Delta\ \Gamma, x_i : \tau_i \vdash^{\mathbb{E}} \grave{e}_i \rightsquigarrow e_i : \tau\}_{i \in L}}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prcase}(\grave{e}; \{i \hookrightarrow x_i.\grave{e}_i\}_{i \in L}) \rightsquigarrow \texttt{case}(e; \{i \hookrightarrow x_i.e_i\}_{i \in L}) : \tau} \tag{B.10m}$$

$$\frac{\begin{array}{c}\emptyset \vdash^{\mathsf{ts}(\mathbb{E})} \grave{\tau} \rightsquigarrow \tau\ \mathsf{type} \qquad \mathbb{E} = \langle \mathcal{D}; \Delta_{\mathrm{app}} \rangle;\ \langle \mathcal{G}; \Gamma_{\mathrm{app}} \rangle;\ \hat{\Psi}; \hat{\Phi};\ b \\ \mathsf{parseUExp}(\mathsf{subseq}(b; m; n)) = \hat{e} \qquad \langle \mathcal{D}; \Delta_{\mathrm{app}} \rangle\ \langle \mathcal{G}; \Gamma_{\mathrm{app}} \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{e} \rightsquigarrow e : \tau \\ \Delta \cap \Delta_{\mathrm{app}} = \emptyset \qquad \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma_{\mathrm{app}}) = \emptyset\end{array}}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{splicede}[m; n; \grave{\tau}] \rightsquigarrow e : \tau} \tag{B.10n}$$

$$\frac{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau \qquad \{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{r}_i \rightsquigarrow r_i : \tau \Mapsto \tau'\}_{1 \le i \le n}}{\Delta\ \Gamma \vdash^{\mathbb{E}} \texttt{prmatch}(\grave{e}; \{\grave{r}_i\}_{1 \le i \le n}) \rightsquigarrow \texttt{match}(e; \{r_i\}_{1 \le i \le n}) : \tau'} \tag{B.10o}$$

$\boxed{\Delta\;\Gamma \vdash^{\mathbb{E}} \grave{r} \rightsquigarrow r : \tau \Mapsto \tau'}$ $\grave{r}$ has expansion r taking values of type τ to values of type τ'

$$\frac{\Delta \cup \Delta_{\text{app}} \vdash p : \tau \dashv\!\vert\, \Gamma' \qquad \Delta\;\Gamma \cup \Gamma' \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau'}{\Delta\;\Gamma \vdash^{\mathbb{E}} \texttt{prrule}(p.\grave{e}) \rightsquigarrow \texttt{rule}(p.e) : \tau \Mapsto \tau'} \tag{B.11}$$

B.4.4 Proto-Pattern Validation

Pattern splicing scenes, $\mathbb{P}$, are of the form $\hat{\Delta}; \hat{\Phi}; b$.

$\boxed{\grave{p} \rightsquigarrow p : \tau \dashv\!\vert^{\mathbb{P}}\, \hat{\Gamma}}$ $\grave{p}$ has expansion p matching against τ generating hypotheses $\hat{\Gamma}$

$$\frac{}{\texttt{prwildp} \rightsquigarrow \texttt{wildp} : \tau \dashv\!\vert^{\mathbb{P}}\, \langle\emptyset;\emptyset\rangle} \tag{B.12a}$$

$$\frac{\grave{p} \rightsquigarrow p : [\texttt{rec}(t.\tau)/t]\tau \dashv\!\vert^{\mathbb{P}}\, \hat{\Gamma}}{\texttt{prfoldp}(\grave{p}) \rightsquigarrow \texttt{foldp}(p) : \texttt{rec}(t.\tau) \dashv\!\vert^{\mathbb{P}}\, \hat{\Gamma}} \tag{B.12b}$$

$$\frac{\begin{array}{c}\tau = \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i\in L})\\ \{\grave{p}_i \rightsquigarrow p_i : \tau_i \dashv\!\vert^{\mathbb{P}}\, \hat{\Gamma}_i\}_{i\in L}\end{array}}{\texttt{prtplp}[L](\{i \hookrightarrow \grave{p}_i\}_{i\in L}) \rightsquigarrow \texttt{tplp}(\{i \hookrightarrow p_i\}_{i\in L}) : \tau \dashv\!\vert^{\mathbb{P}}\, \uplus_{i\in L}\hat{\Gamma}_i} \tag{B.12c}$$

$$\frac{\grave{p} \rightsquigarrow p : \tau \dashv\!\vert^{\mathbb{P}}\, \hat{\Gamma}}{\texttt{prinjp}[\ell](\grave{p}) \rightsquigarrow \texttt{injp}[\ell](p) : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L}; \ell \hookrightarrow \tau) \dashv\!\vert^{\mathbb{P}}\, \hat{\Gamma}} \tag{B.12d}$$

$$\frac{\emptyset \vdash^{\hat{\Delta};b} \grave{\tau} \rightsquigarrow \tau \;\mathsf{type} \qquad \mathsf{parseUPat}(\mathsf{subseq}(b;m;n)) = \hat{p} \qquad \hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \rightsquigarrow p : \tau \dashv\!\vert\, \hat{\Gamma}}{\texttt{splicedp}[m;n;\grave{\tau}] \rightsquigarrow p : \tau \dashv\!\vert^{\hat{\Delta};\hat{\Phi};b}\, \hat{\Gamma}} \tag{B.12e}$$

B.5 Metatheory

B.5.1 Type Expansion

Lemma B.25 (Type Expansion)**.** *If* $\langle\mathcal{D};\Delta\rangle \vdash \hat{\tau} \rightsquigarrow \tau \;\mathsf{type}$ *then* $\Delta \vdash \tau \;\mathsf{type}$.

Proof. By rule induction over Rules (B.5). In each case, we apply the IH to or over each premise, then apply the corresponding type formation rule in Rules (B.1). □

Lemma B.26 (Proto-Type Validation)**.** *If* $\Delta \vdash^{\langle\mathcal{D};\Delta_{app}\rangle;b} \grave{\tau} \rightsquigarrow \tau \;\mathsf{type}$ *and* $\Delta \cap \Delta_{app} = \emptyset$ *then* $\Delta \cup \Delta_{app} \vdash \tau \;\mathsf{type}$.

Proof. By rule induction over Rules (B.9).

Case (B.9a).

(1) $\Delta = \Delta', t \; \mathsf{type}$ — by assumption

(2) $\grave{\tau} = t$ — by assumption

(3) $\tau = t$ — by assumption

(4) $\Delta', t \; \mathsf{type} \vdash t \; \mathsf{type}$ — by Rule (B.1a)

(5) $\Delta', t \; \mathsf{type} \cup \Delta_{\mathsf{app}} \vdash t \; \mathsf{type}$ — by Lemma B.2 over Δ_{app} to (4)

Case (B.9b).

(1) $\grave{\tau} = \mathtt{prparr}(\grave{\tau}_1; \grave{\tau}_2)$ — by assumption

(2) $\tau = \mathtt{parr}(\tau_1; \tau_2)$ — by assumption

(3) $\Delta \vdash^{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle; b} \grave{\tau}_1 \leadsto \tau_1 \; \mathsf{type}$ — by assumption

(4) $\Delta \vdash^{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle; b} \grave{\tau}_2 \leadsto \tau_2 \; \mathsf{type}$ — by assumption

(5) $\Delta \cup \Delta_{\mathsf{app}} \vdash \tau_1 \; \mathsf{type}$ — by IH on (3)

(6) $\Delta \cup \Delta_{\mathsf{app}} \vdash \tau_2 \; \mathsf{type}$ — by IH on (4)

(7) $\Delta \cup \Delta_{\mathsf{app}} \vdash \mathtt{parr}(\tau_1; \tau_2) \; \mathsf{type}$ — by Rule (B.1b) on (5) and (6)

Case (B.9c).

(1) $\grave{\tau} = \mathtt{prall}(t.\grave{\tau}')$ — by assumption

(2) $\tau = \mathtt{all}(t.\tau')$ — by assumption

(3) $\Delta, t \; \mathsf{type} \vdash^{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle; b} \grave{\tau}' \leadsto \tau' \; \mathsf{type}$ — by assumption

(4) $\Delta, t \; \mathsf{type} \cup \Delta_{\mathsf{app}} \vdash \tau' \; \mathsf{type}$ — by IH on (3)

(5) $\Delta \cup \Delta_{\mathsf{app}}, t \; \mathsf{type} \vdash \tau' \; \mathsf{type}$ — by exchange over Δ_{app} on (4)

(6) $\Delta \cup \Delta_{\mathsf{app}} \vdash \mathtt{all}(t.\tau') \; \mathsf{type}$ — by Rule (B.1c) on (5)

Case (B.9d).

(1) $\grave{\tau} = \mathtt{prrec}(t.\grave{\tau}')$ — by assumption

(2) $\tau = \mathtt{rec}(t.\tau')$ — by assumption

(3) $\Delta, t \; \mathsf{type} \vdash^{\Delta_{\mathsf{app}}; b} \grave{\tau}' \leadsto \tau' \; \mathsf{type}$ — by assumption

(4) $\Delta, t \text{ type} \cup \Delta_{\text{app}} \vdash \tau' \text{ type}$ — by IH on (3)

(5) $\Delta \cup \Delta_{\text{app}}, t \text{ type} \vdash \tau' \text{ type}$ — by exchange over Δ_{app} on (4)

(6) $\Delta \cup \Delta_{\text{app}} \vdash \texttt{rec}(t.\tau') \text{ type}$ — by Rule (B.1d) on (5)

Case (B.9e).

(1) $\grave{\tau} = \texttt{prprod}(\{i \hookrightarrow \grave{\tau}_i\}_{i \in L})$ — by assumption

(2) $\tau = \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L})$ — by assumption

(3) $\{\Delta \vdash^{\Delta_{\text{app}}; b} \grave{\tau}_i \rightsquigarrow \tau_i \text{ type}\}_{i \in L}$ — by assumption

(4) $\{\Delta \cup \Delta_{\text{app}} \vdash \tau_i \text{ type}\}_{i \in L}$ — by IH over (3)

(5) $\Delta \cup \Delta_{\text{app}} \vdash \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}) \text{ type}$ — by Rule (B.1e) on (4)

Case (B.9f).

(1) $\grave{\tau} = \texttt{prsum}(\{i \hookrightarrow \grave{\tau}_i\}_{i \in L})$ — by assumption

(2) $\tau = \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L})$ — by assumption

(3) $\{\Delta \vdash^{\Delta_{\text{app}}; b} \grave{\tau}_i \rightsquigarrow \tau_i \text{ type}\}_{i \in L}$ — by assumption

(4) $\{\Delta \cup \Delta_{\text{app}} \vdash \tau_i \text{ type}\}_{i \in L}$ — by IH over (3)

(5) $\Delta \cup \Delta_{\text{app}} \vdash \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}) \text{ type}$ — by Rule (B.1f) on (4)

Case (B.9g).

(1) $\grave{\tau} = \texttt{splicedt}[m; n]$ — by assumption

(2) $\mathsf{parseUTyp}(\mathsf{subseq}(b; m; n)) = \hat{\tau}$ — by assumption

(3) $\langle \mathcal{D}; \Delta_{\text{app}} \rangle \vdash \hat{\tau} \rightsquigarrow \tau \text{ type}$ — by assumption

(4) $\Delta \cap \Delta_{\text{app}} = \emptyset$ — by assumption

(5) $\Delta_{\text{app}} \vdash \tau \text{ type}$ — by Lemma B.25 on (3)

(6) $\Delta \cup \Delta_{\text{app}} \vdash \tau \text{ type}$ — by Lemma B.2 over Δ on (5) and exchange over Δ

$\square$

B.5.2 Typed Pattern Expansion

Theorem B.27 (Typed Pattern Expansion).

1. *If* $\langle \mathcal{D}; \Delta \rangle \vdash_{\langle \mathcal{A}; \Phi \rangle} \hat{p} \rightsquigarrow p : \tau \dashv\!\mid \langle \mathcal{G}; \Gamma \rangle$ *then* $p : \tau \dashv\!\mid \Gamma$.

2. *If* $\grave{p} \rightsquigarrow p : \tau \dashv\!\mid^{\langle \mathcal{D};\Delta \rangle; \langle \mathcal{A};\Phi \rangle; b} \langle \mathcal{G}; \Gamma \rangle$ *then* $p : \tau \dashv\!\mid \Gamma$.

Proof. By mutual rule induction over Rules (B.8) and Rules (B.12).

1. We induct on the premise. In the following, let $\hat{\Delta} = \langle \mathcal{D}; \Delta \rangle$ and $\hat{\Gamma} = \langle \mathcal{G}; \Gamma \rangle$ and $\hat{\Phi} = \langle \mathcal{A}; \Phi \rangle$.

   Case (B.8a).

(1) $\hat{p} = \hat{x}$	by assumption
(2) $p = x$	by assumption
(3) $\Gamma = x : \tau$	by assumption
(4) $x : \tau \dashv\!\mid x : \tau$	by Rule (B.4a)

   Case (B.8b).

(1) $p = \texttt{wildp}$	by assumption
(2) $\Gamma = \emptyset$	by assumption
(3) $\texttt{wildp} : \tau \dashv\!\mid \emptyset$	by Rule (B.4b)

   Case (B.8c).

(1) $\hat{p} = \texttt{fold}(\hat{p}')$	by assumption
(2) $p = \texttt{foldp}(p')$	by assumption
(3) $\tau = \texttt{rec}(t.\tau')$	by assumption
(4) $\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p}' \rightsquigarrow p' : [\texttt{rec}(t.\tau')/t]\tau' \dashv\!\mid \hat{\Gamma}$	by assumption
(5) $p' : [\texttt{rec}(t.\tau')/t]\tau' \dashv\!\mid \Gamma$	by IH, part 1 on (4)
(6) $\texttt{foldp}(p') : \texttt{rec}(t.\tau') \dashv\!\mid \Gamma$	by Rule (B.4c) on (5)

   Case (B.8d).

(1) $\hat{p} = \langle \{i \hookrightarrow \hat{p}_i\}_{i \in L} \rangle$	by assumption
(2) $p = \texttt{tplp}(\{i \hookrightarrow p_i\}_{i \in L})$	by assumption
(3) $\tau = \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L})$	by assumption
(4) $\{\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p}_i \rightsquigarrow p_i : \tau_i \dashv\!\mid \langle \mathcal{G}_i; \Gamma_i \rangle\}_{i \in L}$	by assumption
(5) $\Gamma = \cup_{i \in L} \Gamma_i$	by assumption
(6) $\{p_i : \tau_i \dashv\!\mid \Gamma_i\}_{i \in L}$	by IH, part 1 over (4)
(7) $\texttt{tplp}(\{i \hookrightarrow p_i\}_{i \in L}) : \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}) \dashv\!\mid \cup_{i \in L} \Gamma_i$	by Rule (B.4d) on (6)

Case (B.8e).

(1) $\hat{p} = \mathtt{inj}[\ell](\hat{p}')$	by assumption
(2) $p = \mathtt{injp}[\ell](p')$	by assumption
(3) $\tau = \mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau')$	by assumption
(4) $\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p}' \rightsquigarrow p' : \tau' \dashv\!\!\mid \hat{\Gamma}$	by assumption
(5) $p' : \tau' \dashv\!\!\mid \Gamma$	by IH, part 1 on (4)
(6) $\mathtt{injp}[\ell](p') : \mathtt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau') \dashv\!\!\mid \Gamma$	by Rule (B.4e) on (5)

Case (B.8f).

(1) $\hat{p} = \hat{a}\ \mathtt{`}(b)\mathtt{`}$	by assumption
(2) $\mathcal{A} = \mathcal{A}', \hat{a} \rightsquigarrow x$	by assumption
(3) $\Phi = \Phi', a \hookrightarrow \mathtt{sptlm}(\tau; e_{\text{parse}})$	by assumption
(4) $b \downarrow_{\mathsf{Body}} e_{\text{body}}$	by assumption
(5) $e_{\text{parse}}(e_{\text{body}}) \Downarrow \mathtt{inj}[\mathtt{SuccessP}](e_{\text{proto}})$	by assumption
(6) $e_{\text{proto}} \uparrow_{\mathsf{PrPat}} \grave{p}$	by assumption
(7) $\grave{p} \rightsquigarrow p : \tau \dashv\!\!\mid^{\hat{\Delta}; \langle \mathcal{A}; \Phi \rangle; b} \langle \mathcal{G}; \Gamma \rangle$	by assumption
(8) $p : \tau \dashv\!\!\mid \Gamma$	by IH, part 2 on (7)

2. We induct on the premise. In the following, let $\hat{\Gamma} = \langle \mathcal{G}; \Gamma \rangle$ and $\hat{\Delta} = \langle \mathcal{D}; \Delta \rangle$ and $\hat{\Phi} = \langle \mathcal{A}; \Phi \rangle$.

Case (B.12a).

(1) $p = \mathtt{wildp}$	by assumption
(2) $\Gamma = \emptyset$	by assumption
(3) $\mathtt{wildp} : \tau \dashv\!\!\mid \emptyset$	by Rule (B.4b)

Case (B.12b).

(1) $\grave{p} = \mathtt{prfoldp}(\grave{p}')$	by assumption
(2) $p = \mathtt{foldp}(p')$	by assumption
(3) $\tau = \mathtt{rec}(t.\tau')$	by assumption
(4) $\grave{p}' \rightsquigarrow p' : [\mathtt{rec}(t.\tau')/t]\tau' \dashv\!\!\mid^{\hat{\Delta}; \hat{\Phi}; b} \hat{\Gamma}$	by assumption
(5) $p' : [\mathtt{rec}(t.\tau')/t]\tau' \dashv\!\!\mid \Gamma$	by IH, part 2 on (4)
(6) $\mathtt{foldp}(p') : \mathtt{rec}(t.\tau') \dashv\!\!\mid \Gamma$	by Rule (B.4c) on (5)

Case (B.12c).

(1) $\grave{p} = \mathtt{prtplp}[L](\{i \hookrightarrow \grave{p}_i\}_{i \in L})$	by assumption
(2) $p = \mathtt{tplp}(\{i \hookrightarrow p_i\}_{i \in L})$	by assumption
(3) $\tau = \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L})$	by assumption

(4) $\{\grave{p}_i \rightsquigarrow p_i : \tau_i \dashv\!\!\mid^{\hat{\Delta};\hat{\Phi};b} \langle \mathcal{G}_i; \Gamma_i \rangle\}_{i \in L}$ by assumption

(5) $\Gamma = \cup_{i \in L} \Gamma_i$ by assumption

(6) $\{p_i : \tau_i \dashv\!\!\mid \Gamma_i\}_{i \in L}$ by IH, part 2 over (4)

(7) $\texttt{tplp}(\{i \hookrightarrow p_i\}_{i \in L}) : \texttt{prod}(\{i \hookrightarrow \tau_i\}_{i \in L}) \dashv\!\!\mid \cup_{i \in L} \Gamma_i$
by Rule (B.4d) on (6)

Case (B.12d).

(1) $\grave{p} = \texttt{prinjp}[\ell](\grave{p}')$ by assumption

(2) $p = \texttt{injp}[\ell](p')$ by assumption

(3) $\tau = \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau')$ by assumption

(4) $\grave{p}' \rightsquigarrow p' : \tau' \dashv\!\!\mid^{\hat{\Delta};\hat{\Phi};b} \hat{\Gamma}$ by assumption

(5) $p' : \tau' \dashv\!\!\mid \Gamma$ by IH, part 2 on (4)

(6) $\texttt{injp}[\ell](p') : \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i \in L}; \ell \hookrightarrow \tau') \dashv\!\!\mid \Gamma$ by Rule (B.4e) on (5)

Case (B.12e).

(1) $\grave{p} = \texttt{splicedp}[m; n; \grave{\tau}]$ by assumption

(2) $\emptyset \vdash^{\hat{\Delta}; b} \grave{\tau} \rightsquigarrow \tau \text{ type}$ by assumption

(3) $\mathsf{parseUExp}(\mathsf{subseq}(b; m; n)) = \hat{p}$ by assumption

(4) $\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \rightsquigarrow p : \tau \dashv\!\!\mid \hat{\Gamma}$ by assumption

(5) $p : \tau \dashv\!\!\mid \Gamma$ by IH, part 1 on (4)

The mutual induction can be shown to be well-founded by showing that the following numeric metric on the judgements that we induct on is decreasing:

$$\|\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \rightsquigarrow p : \tau \dashv\!\!\mid \hat{\Gamma}\| = \|\hat{p}\|$$
$$\|\grave{p} \rightsquigarrow p : \tau \dashv\!\!\mid^{\hat{\Delta};\hat{\Phi};b} \hat{\Gamma}\| = \|b\|$$

where $\|b\|$ is the length of b and $\|\hat{p}\|$ is the sum of the lengths of the literal bodies in $\hat{p}$, as defined in Sec. B.3.1.

The only case in the proof of part 1 that invokes part 2 is Case (B.8f). There, we have that the metric remains stable:

$$\begin{aligned} &\|\hat{\Delta} \vdash_{\hat{\Phi}} \hat{a}\ \texttt{`(}b\texttt{)`} \rightsquigarrow p : \tau \dashv\!\!\mid \hat{\Gamma}\| \\ =&\|\grave{p} \rightsquigarrow p : \tau \dashv\!\!\mid^{\hat{\Delta};\hat{\Phi};b} \hat{\Gamma}\| \\ =&\|b\| \end{aligned}$$

The only case in the proof of part 2 that invokes part 1 is Case (B.12e). There, we have that $\mathsf{parseUPat}(\mathsf{subseq}(b; m; n)) = \hat{p}$ and the IH is applied to the judgement

$\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \rightsquigarrow p : \tau \dashv\!\mid \hat{\Gamma}$. Because the metric is stable when passing from part 1 to part 2, we must have that it is strictly decreasing in the other direction:

$$\|\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \rightsquigarrow p : \tau \dashv\!\mid \hat{\Gamma}\| < \|\texttt{splicedp}[m; n; \grave{\tau}] \rightsquigarrow p : \tau \dashv\!\mid^{\hat{\Delta}; \hat{\Phi}; b} \hat{\Gamma}\|$$

i.e. by the definitions above,

$$\|\hat{p}\| < \|b\|$$

This is established by appeal to Condition B.17, which states that subsequences of b are no longer than b, and the Condition B.13, which states that an unexpanded pattern constructed by parsing a textual sequence b is strictly smaller, as measured by the metric defined above, than the length of b, because some characters must necessarily be used to apply the pattern TLM and delimit each literal body. Combining Conditions B.17 and B.13, we have that $\|\hat{p}\| < \|b\|$ as needed. □

B.5.3 Typed Expression Expansion

Theorem B.28 (Typed Expansion (Strong)).

1. (a) *If* $\langle \mathcal{D}; \Delta \rangle\ \langle \mathcal{G}; \Gamma \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{e} \rightsquigarrow e : \tau$ *then* $\Delta\ \Gamma \vdash e : \tau$.

   (b) *If* $\langle \mathcal{D}; \Delta \rangle\ \langle \mathcal{G}; \Gamma \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{r} \rightsquigarrow r : \tau \Rrightarrow \tau'$ *then* $\Delta\ \Gamma \vdash r : \tau \Rrightarrow \tau'$.

2. (a) *If* $\Delta\ \Gamma \vdash^{\langle \mathcal{D}; \Delta_{app} \rangle; \langle \mathcal{G}; \Gamma_{app} \rangle; \hat{\Psi}; \hat{\Phi}; b} \grave{e} \rightsquigarrow e : \tau$ *and* $\Delta \cap \Delta_{app} = \emptyset$ *and* $dom(\Gamma) \cap dom(\Gamma_{app}) = \emptyset$ *then* $\Delta \cup \Delta_{app}\ \Gamma \cup \Gamma_{app} \vdash e : \tau$.

   (b) *If* $\Delta\ \Gamma \vdash^{\langle \mathcal{D}; \Delta_{app} \rangle; \langle \mathcal{G}; \Gamma_{app} \rangle; \hat{\Psi}; \hat{\Phi}; b} \grave{r} \rightsquigarrow r : \tau \Rrightarrow \tau'$ *and* $\Delta \cap \Delta_{app} = \emptyset$ *and* $dom(\Gamma) \cap dom(\Gamma_{app}) = \emptyset$ *then* $\Delta \cup \Delta_{app}\ \Gamma \cup \Gamma_{app} \vdash r : \tau \Rrightarrow \tau'$.

Proof. By mutual rule induction over Rules (B.6), Rule (B.7), Rules (B.10) and Rule (B.11).

1. In the following, let $\hat{\Delta} = \langle \mathcal{D}; \Delta \rangle$ and $\hat{\Gamma} = \langle \mathcal{G}; \Gamma \rangle$.

   (a) **Case** (B.6a).

(1) $\hat{e} = \hat{x}$	by assumption
(2) $e = x$	by assumption
(3) $\Gamma = \Gamma', x : \tau$	by assumption
(4) $\Delta\ \Gamma', x : \tau \vdash x : \tau$	by Rule (B.2a)

   Case (B.6b).

(1) $\hat{e} = \hat{e}' : \hat{\tau}$	by assumption
(2) $\hat{\Delta} \vdash \hat{\tau} \rightsquigarrow \tau$ type	by assumption
(3) $\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{e}' \rightsquigarrow e : \tau$	by assumption
(4) $\Delta\ \Gamma \vdash e : \tau$	by IH, part 1(a) on (3)

Case (B.6c).

(1) $\hat{e} = \texttt{let val}\ \hat{x} = \hat{e}_1\ \texttt{in}\ \hat{e}_2$ — by assumption

(2) $e = \texttt{ap(lam\{}\tau_1\texttt{\}(}x.e_2\texttt{);}e_1\texttt{)}$ — by assumption

(3) $\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_1 \rightsquigarrow e_1 : \tau_1$ — by assumption

(4) $\hat{\Delta}\ \hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau_1 \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_2 \rightsquigarrow e_2 : \tau$ — by assumption

(5) $\Delta\ \Gamma \vdash e_1 : \tau_1$ — by IH, part 1(a) on (3)

(6) $\Delta\ \Gamma, x : \tau \vdash e_2 : \tau$ — by IH, part 1(a) on (4)

(7) $\Delta\ \Gamma \vdash \texttt{lam\{}\tau_1\texttt{\}(}x.e_2\texttt{)} : \texttt{parr(}\tau_1\texttt{;}\tau\texttt{)}$ — by Rule (B.2b) on (6)

(8) $\Delta\ \Gamma \vdash \texttt{ap(lam\{}\tau_1\texttt{\}(}x.e_2\texttt{);}e_1\texttt{)} : \tau$ — by Rule (B.2c) on (7) and (5)

Case (B.6d).

(1) $\hat{e} = \lambda\hat{x}{:}\hat{\tau}_1.\hat{e}'$ — by assumption

(2) $e = \texttt{lam\{}\tau_1\texttt{\}(}x.e'\texttt{)}$ — by assumption

(3) $\tau = \texttt{parr(}\tau_1\texttt{;}\tau_2\texttt{)}$ — by assumption

(4) $\hat{\Delta} \vdash \hat{\tau}_1 \rightsquigarrow \tau_1\ \mathsf{type}$ — by assumption

(5) $\hat{\Delta}\ \hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau_1 \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}' \rightsquigarrow e' : \tau_2$ — by assumption

(6) $\Delta \vdash \tau_1\ \mathsf{type}$ — by Lemma B.25 on (4)

(7) $\Delta\ \Gamma, x : \tau_1 \vdash e' : \tau_2$ — by IH, part 1(a) on (5)

(8) $\Delta\ \Gamma \vdash \texttt{lam\{}\tau_1\texttt{\}(}x.e'\texttt{)} : \texttt{parr(}\tau_1\texttt{;}\tau_2\texttt{)}$ — by Rule (B.2b) on (6) and (7)

Case (B.6e).

(1) $\hat{e} = \hat{e}_1(\hat{e}_2)$ — by assumption

(2) $e = \texttt{ap(}e_1\texttt{;}e_2\texttt{)}$ — by assumption

(3) $\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_1 \rightsquigarrow e_1 : \texttt{parr(}\tau_2\texttt{;}\tau\texttt{)}$ — by assumption

(4) $\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_2 \rightsquigarrow e_2 : \tau_2$ — by assumption

(5) $\Delta\ \Gamma \vdash e_1 : \texttt{parr(}\tau_2\texttt{;}\tau\texttt{)}$ — by IH, part 1(a) on (3)

(6) $\Delta\ \Gamma \vdash e_2 : \tau_2$ — by IH, part 1(a) on (4)

(7) $\Delta\ \Gamma \vdash \texttt{ap(}e_1\texttt{;}e_2\texttt{)} : \tau$ — by Rule (B.2c) on (5) and (6)

Case (B.6f) **through** (B.6m)**.** These cases follow analagously, i.e. we apply Lemma B.25 to or over the type expansion premises and the IH part 1(a) to or over the typed expression expansion premises and then apply the corresponding typing rule in Rules (B.2d) through (B.2k).

Case (B.6n).

(1) $\hat{e} =$
$\texttt{notation}\ \hat{a}\ \texttt{at}\ \hat{\tau}'\ \{\ \texttt{expr parser}\ e_{\text{parse}};\texttt{expansions require}\ \hat{e}_{\text{dep}}\ \}\ \texttt{in}\ \hat{e}'$ by assumption

(2) $\hat{\Delta} \vdash \hat{\tau}' \leadsto \tau'\ \mathsf{type}$ by assumption

(3) $\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e}_{\text{dep}} \leadsto e_{\text{dep}} : \tau_{\text{dep}}$ by assumption

(4) $\emptyset\ \emptyset \vdash e_{\text{parse}} : \texttt{parr(Body;ParseResultE)}$ by assumption

(5) $\hat{\Delta}\ \langle \mathcal{G}; \Gamma, x : \tau_{\text{dep}} \rangle \vdash_{\hat{\Psi}, \hat{a} \leadsto x \hookrightarrow \texttt{setlm}(\tau'; e_{\text{parse}}); \hat{\Phi}} \hat{e}' \leadsto e' : \tau$ by assumption

(6) $\Delta \vdash \tau'\ \mathsf{type}$ by Lemma B.25 to (2)

(7) $\Delta \vdash \tau_{\text{dep}}\ \mathsf{type}$ by Lemma B.25 to (3)

(8) $\Delta\ \Gamma, x : \tau_{\text{dep}} \vdash e' : \tau$ by IH, part 1(a) on (5)

(9) $\Delta\ \Gamma \vdash e_{\text{dep}} : \tau_{\text{dep}}$ by IH, part 1(a) on (3)

(10) $e = \texttt{ap(lam}\{\tau_{\text{dep}}\}(x.e');e_{\text{dep}})$ by assumption

(11) $\Delta\ \Gamma \vdash e : \tau$ by Rule (B.2c) and Rule (B.2b) with (8) and (7) and (9)

Case (B.6o).

(1) $\hat{e} = \hat{a}\ \text{`}(b)\text{`}$ by assumption

(2) $\mathcal{A} = \mathcal{A}', \hat{a} \leadsto x$ by assumption

(3) $\Psi = \Psi', x \hookrightarrow \texttt{setlm}(\tau; e_{\text{parse}})$ by assumption

(4) $\Gamma = \Gamma', x : \tau_{\text{dep}}$ by assumption

(5) $e = \texttt{ap}(e'; x)$ by assumption

(6) $b \downarrow_{\mathsf{Body}} e_{\text{body}}$ by assumption

(7) $e_{\text{parse}}(e_{\text{body}}) \Downarrow \texttt{inj[SuccessE]}(e_{\text{proto}})$ by assumption

(8) $e_{\text{proto}} \uparrow_{\mathsf{PrExpr}} \grave{e}$ by assumption

(9) $\emptyset\ \emptyset \vdash^{\hat{\Delta};\hat{\Gamma};\hat{\Psi};\hat{\Phi};b} \grave{e} \leadsto e' : \texttt{parr}(\tau_{\text{dep}}; \tau)$ by assumption

(10) $\emptyset \cap \Delta = \emptyset$ by finite set intersection

(11) $\emptyset \cap \text{dom}(\Gamma) = \emptyset$ by finite set intersection

(12) $\emptyset \cup \Delta\ \emptyset \cup \Gamma \vdash e' : \texttt{parr}(\tau_{\text{dep}}; \tau)$ by IH, part 2(a) on (9), (10), and (11)

(13) $\Delta\ \Gamma \vdash e' : \texttt{parr}(\tau_{\text{dep}}; \tau)$ by finite set and finite function identity over (12)

(14) $\Delta\ \Gamma \vdash x : \tau_{\text{dep}}$ by Rule (B.2a)

(15) $\Delta\ \Gamma \vdash e : \tau$ by Rule (B.2c) on (13) and (14)

Case (B.6p).

(1) $\hat{e} = \texttt{match}\ \hat{e}'\ \{\hat{r}_i\}_{1 \le i \le n}$ by assumption
(2) $e = \texttt{match}(e'; \{r_i\}_{1 \le i \le n})$ by assumption
(3) $\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{e}' \rightsquigarrow e' : \tau'$ by assumption
(4) $\{\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{r}_i \rightsquigarrow r_i : \tau' \Mapsto \tau\}_{1 \le i \le n}$ by assumption
(5) $\Delta\ \Gamma \vdash e' : \tau'$ by IH, part 1(a) on (3)
(6) $\{\Delta\ \Gamma \vdash r_i : \tau' \Mapsto \tau\}_{1 \le i \le n}$ by IH, part 1(b) over (4)
(7) $\Delta\ \Gamma \vdash \texttt{match}(e'; \{r_i\}_{1 \le i \le n}) : \tau$ by Rule (B.2l) on (5) and (6)

Case (B.6q).

(1) $\hat{e} = \texttt{notation}\ \hat{a}\ \texttt{at}\ \hat{\tau}'\ \{\ \texttt{pat parser}\ e_{\text{parse}}\ \}\ \texttt{in}\ \hat{e}'$
by assumption
(2) $\hat{\Delta} \vdash \hat{\tau}' \rightsquigarrow \tau'\ \mathsf{type}$ by assumption
(3) $\emptyset\ \emptyset \vdash e_{\text{parse}} : \texttt{parr(Body;ParseResultE)}$ by assumption
(4) $\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi}; \hat{\Phi}, \hat{a} \rightsquigarrow x \hookrightarrow \texttt{sptlm}(\tau'; e_{\text{parse}})} \hat{e}' \rightsquigarrow e : \tau$ by assumption
(5) $\Delta \vdash \tau'\ \mathsf{type}$ by Lemma B.25 to (2)
(6) $\Delta\ \Gamma \vdash e : \tau$ by IH, part 1(a) on (4)

(b) **Case** (B.7).

(1) $\hat{r} = \hat{p} \Rightarrow \hat{e}$ by assumption
(2) $r = \texttt{rule}(p.e)$ by assumption
(3) $\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \rightsquigarrow p : \tau \dashv\!\!\mid \langle \mathcal{A}'; \Gamma \rangle$ by assumption
(4) $\hat{\Delta}\ \langle \mathcal{A} \uplus \mathcal{A}'; \Gamma \cup \Gamma \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{e} \rightsquigarrow e : \tau'$ by assumption
(5) $p : \tau \dashv\!\!\mid \Gamma$ by Theorem B.27, part 1 on (3)
(6) $\Delta\ \Gamma \cup \Gamma \vdash e : \tau'$ by IH, part 1(a) on (4)
(7) $\Delta\ \Gamma \vdash \texttt{rule}(p.e) : \tau \Mapsto \tau'$ by Rule (B.3) on (5) and (6)

2. In the following, let $\hat{\Delta} = \langle \mathcal{D}; \Delta_{\text{app}} \rangle$ and $\hat{\Gamma} = \langle \mathcal{G}; \Gamma_{\text{app}} \rangle$.

(a) **Case** (B.10a).

(1) $\grave{e} = x$ by assumption
(2) $e = x$ by assumption
(3) $\Gamma = \Gamma', x : \tau$ by assumption
(4) $\Delta \cup \Delta_{\text{app}}\ \Gamma', x : \tau \vdash x : \tau$ by Rule (B.2a)

(5) $\Delta \cup \Delta_{\text{app}}\ \Gamma', x : \tau \cup \Gamma_{\text{app}} \vdash x : \tau$ — by Lemma B.2 over Γ_{app} to (4)

Case (B.10d).

(1) $\grave{e} = \texttt{prlam}\{\grave{\tau}_1\}(x.\grave{e}')$ — by assumption

(2) $e = \texttt{lam}\{\tau_1\}(x.e')$ — by assumption

(3) $\tau = \texttt{parr}(\tau_1; \tau_2)$ — by assumption

(4) $\Delta \vdash^{\hat{\Delta}_{\text{app}}; b} \grave{\tau}_1 \rightsquigarrow \tau_1$ type — by assumption

(5) $\Delta\ \Gamma, x : \tau_1 \vdash^{\hat{\Delta}_{\text{app}}; \hat{\Gamma}_{\text{app}}; \hat{\Psi}; \hat{\Phi}; b} \grave{e}' \rightsquigarrow e' : \tau_2$ — by assumption

(6) $\Delta \cap \Delta_{\text{app}} = \emptyset$ — by assumption

(7) $\text{dom}(\Gamma) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset$ — by assumption

(8) $x \notin \text{dom}(\Gamma_{\text{app}})$ — by identification convention

(9) $\text{dom}(\Gamma, x : \tau_1) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset$ — by (7) and (8)

(10) $\Delta \cup \Delta_{\text{app}} \vdash \tau_1$ type — by Lemma B.26 on (4) and (6)

(11) $\Delta \cup \Delta_{\text{app}}\ \Gamma, x : \tau_1 \cup \Gamma_{\text{app}} \vdash e' : \tau_2$ — by IH, part 2(a) on (5), (6) and (9)

(12) $\Delta \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}}, x : \tau_1 \vdash e' : \tau_2$ — by exchange over Γ_{app} on (11)

(13) $\Delta \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}} \vdash \texttt{lam}\{\tau_1\}(x.e') : \texttt{parr}(\tau_1; \tau_2)$ — by Rule (B.2b) on (10) and (12)

Case (B.10e).

(1) $\grave{e} = \texttt{prap}(\grave{e}_1; \grave{e}_2)$ — by assumption

(2) $e = \texttt{ap}(e_1; e_2)$ — by assumption

(3) $\Delta\ \Gamma \vdash^{\hat{\Delta}_{\text{app}}; \hat{\Gamma}_{\text{app}}; \hat{\Psi}; \hat{\Phi}; b} \grave{e}_1 \rightsquigarrow e_1 : \texttt{parr}(\tau_2; \tau)$ — by assumption

(4) $\Delta\ \Gamma \vdash^{\hat{\Delta}_{\text{app}}; \hat{\Gamma}_{\text{app}}; \hat{\Psi}; \hat{\Phi}; b} \grave{e}_2 \rightsquigarrow e_2 : \tau_2$ — by assumption

(5) $\Delta \cap \Delta_{\text{app}} = \emptyset$ — by assumption

(6) $\text{dom}(\Gamma) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset$ — by assumption

(7) $\Delta \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}} \vdash e_1 : \texttt{parr}(\tau_2; \tau)$ — by IH, part 2(a) on (3), (5) and (6)

(8) $\Delta \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}} \vdash e_2 : \tau_2$ — by IH, part 2(a) on (4), (5) and (6)

(9) $\Delta \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}} \vdash \texttt{ap}(e_1; e_2) : \tau$ — by Rule (B.2c) on (7) and (8)

Case (B.10f).

(1) $\grave{e} = \texttt{prtlam}(t.\grave{e}')$ — by assumption

(2) $e = \texttt{tlam}(t.e')$ — by assumption

(3) $\tau = \mathtt{all}(t.\tau')$	by assumption
(4) $\Delta, t\ \mathsf{type}\ \Gamma \vdash^{\hat{\Delta}_{\text{app}};\hat{\Gamma}_{\text{app}};\hat{\Psi};\hat{\Phi};b} \grave{e}' \rightsquigarrow e' : \tau'$	by assumption
(5) $\Delta \cap \Delta_{\text{app}} = \emptyset$	by assumption
(6) $\text{dom}(\Gamma) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset$	by assumption
(7) $t\ \mathsf{type} \notin \Delta_{\text{app}}$	by identification convention
(8) $\Delta, t\ \mathsf{type} \cap \Delta_{\text{app}} = \emptyset$	by (5) and (7)
(9) $\Delta, t\ \mathsf{type} \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}} \vdash e' : \tau'$	by IH, part 2(a) on (4), (8) and (6)
(10) $\Delta \cup \Delta_{\text{app}}, t\ \mathsf{type}\ \Gamma \cup \Gamma_{\text{app}} \vdash e' : \tau'$	by exchange over Δ_{app} on (9)
(11) $\Delta \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}} \vdash \mathtt{tlam}(t.e') : \mathtt{all}(t.\tau')$	by Rule (B.2d) on (10)

Case (B.10g) **through** (B.10m). These cases follow analagously, i.e. we apply the IH, part 2(a) to all proto-expression validation judgements, Lemma B.26 to all proto-type validation judgements, the identification convention to ensure that extended contexts remain disjoint, weakening and exchange as needed, and the corresponding typing rule in Rules (B.2e) through (B.2k).

Case (B.10n).

(1) $\grave{e} = \mathtt{splicede}[m;n;\grave{\tau}]$	by assumption
(2) $\mathbb{E} = \langle \mathcal{D}; \Delta_{\text{app}} \rangle;\ \langle \mathcal{G}; \Gamma_{\text{app}} \rangle;\ \hat{\Psi};\ b$	by assumption
(3) $\emptyset \vdash^{\mathsf{ts}(\mathbb{E})} \grave{\tau} \rightsquigarrow \tau\ \mathsf{type}$	by assumption
(4) $\mathsf{parseUExp}(\mathsf{subseq}(b;m;n)) = \hat{e}$	by assumption
(5) $\hat{\Delta}_{\text{app}}\ \hat{\Gamma}_{\text{app}} \vdash_{\hat{\Psi}} \hat{e} \rightsquigarrow e : \tau$	by assumption
(6) $\Delta \cap \Delta_{\text{app}} = \emptyset$	by assumption
(7) $\text{dom}(\Gamma) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset$	by assumption
(8) $\Delta_{\text{app}}\ \Gamma_{\text{app}} \vdash e : \tau$	by IH, part 1 on (5)
(9) $\Delta \cup \Delta_{\text{app}}\ \Gamma \cup \Gamma_{\text{app}} \vdash e : \tau$	by Lemma B.2 over Δ and Γ and exchange on (8)

Case (B.10o).

(1) $\grave{e} = \mathtt{prmatch}(\grave{e}'; \{\grave{r}_i\}_{1 \le i \le n})$	by assumption
(2) $e = \mathtt{match}(e'; \{r_i\}_{1 \le i \le n})$	by assumption
(3) $\Delta\ \Gamma \vdash^{\hat{\Delta};\hat{\Gamma};\hat{\Psi};\hat{\Phi};b} \grave{e}' \rightsquigarrow e' : \tau'$	by assumption
(4) $\{\Delta\ \Gamma \vdash^{\hat{\Delta};\hat{\Gamma};\hat{\Psi};\hat{\Phi};b} \grave{r}_i \rightsquigarrow r_i : \tau' \Mapsto \tau\}_{1 \le i \le n}$	by assumption
(5) $\Delta \cap \Delta_{\text{app}} = \emptyset$	by assumption

(6) $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Gamma_{\mathsf{app}}) = \emptyset$ by assumption

(7) $\Delta \cup \Delta_{\mathsf{app}}\ \Gamma \cup \Gamma_{\mathsf{app}} \vdash e' : \tau'$ by IH, part 2(a) on (3), (5) and (6)

(8) $\Delta \cup \Delta_{\mathsf{app}}\ \Gamma \cup \Gamma_{\mathsf{app}} \vdash r : \tau' \Rrightarrow \tau$ by IH, part 2(b) on (4), (5) and (6)

(9) $\Delta \cup \Delta_{\mathsf{app}}\ \Gamma \cup \Gamma_{\mathsf{app}} \vdash \mathtt{match}(e'; \{r_i\}_{1 \le i \le n}) : \tau$ by Rule (B.2l) on (7) and (8)

(b) There is only one case.

Case (B.11).

(1) $\grave{r} = \mathtt{prrule}(p.\grave{e})$ by assumption

(2) $r = \mathtt{rule}(p.e)$ by assumption

(3) $p : \tau \dashv\!\!| \Gamma'$ by assumption

(4) $\Delta\ \Gamma \cup \Gamma' \vdash^{\hat{\Delta};\hat{\Gamma};\hat{\Psi};\hat{\Phi};b} \grave{e} \rightsquigarrow e : \tau'$ by assumption

(5) $\Delta \cap \Delta_{\mathsf{app}} = \emptyset$ by assumption

(6) $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Gamma') = \emptyset$ by identification convention

(7) $\mathsf{dom}(\Gamma_{\mathsf{app}}) \cap \mathsf{dom}(\Gamma') = \emptyset$ by identification convention

(8) $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Gamma_{\mathsf{app}}) = \emptyset$ by assumption

(9) $\mathsf{dom}(\Gamma \cup \Gamma') \cap \mathsf{dom}(\Gamma_{\mathsf{app}}) = \emptyset$ by standard finite set definitions and identities on (6), (7) and (8)

(10) $\Delta \cup \Delta_{\mathsf{app}}\ \Gamma \cup \Gamma' \cup \Gamma_{\mathsf{app}} \vdash e : \tau'$ by IH, part 2(a) on (4), (5) and (9)

(11) $\Delta \cup \Delta_{\mathsf{app}}\ \Gamma \cup \Gamma_{\mathsf{app}} \cup \Gamma' \vdash e : \tau'$ by exchange of Γ' and Γ_{app} on (10)

(12) $\Delta \cup \Delta_{\mathsf{app}}\ \Gamma \cup \Gamma_{\mathsf{app}} \vdash \mathtt{rule}(p.e) : \tau \Rrightarrow \tau'$ by Rule (B.3) on (3) and (11)

The mutual induction can be shown to be well-founded by showing that the following numeric metric on the judgements that we induct on is decreasing:

$$\|\hat{\Delta}\ \hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau\| = \|\hat{e}\|$$

$$\|\Delta\ \Gamma \vdash^{\hat{\Delta};\hat{\Gamma};\hat{\Psi};\hat{\Phi};b} \grave{e} \rightsquigarrow e : \tau\| = \|b\|$$

where $\|b\|$ is the length of b and $\|\hat{e}\|$ is the sum of the lengths of the seTLM literal bodies in $\hat{e}$, as defined in Sec. B.3.1.

The only case in the proof of part 1 that invokes part 2 is Case (B.6o). There, we have that the metric remains stable:

$$\begin{aligned}
&\|\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{a}\;\text{`}(b)\text{`} \rightsquigarrow e : \tau\| \\
=&\|\emptyset\;\emptyset \vdash^{\hat{\Delta};\hat{\Gamma};\hat{\Psi};\hat{\Phi};b} \grave{e} \rightsquigarrow e : \tau\| \\
=&\|b\|
\end{aligned}$$

The only case in the proof of part 2 that invokes part 1 is Case (B.10n). There, we have that $\mathsf{parseUExp}(\mathsf{subseq}(b; m; n)) = \hat{e}$ and the IH is applied to the judgement $\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau$. Because the metric is stable when passing from part 1 to part 2, we must have that it is strictly decreasing in the other direction:

$$\|\hat{\Delta}\;\hat{\Gamma} \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau\| < \|\Delta\;\Gamma \vdash^{\hat{\Delta};\hat{\Gamma};\hat{\Psi};\hat{\Phi};b} \mathtt{splicede}[m; n; \grave{\tau}] \rightsquigarrow e : \tau\|$$

i.e. by the definitions above,

$$\|\hat{e}\| < \|b\|$$

This is established by appeal to Condition B.17, which states that subsequences of b are no longer than b, and Condition B.12, which states that an unexpanded expression constructed by parsing a textual sequence b is strictly smaller, as measured by the metric defined above, than the length of b, because some characters must necessarily be used to apply a TLM and delimit each literal body. Combining these conditions, we have that $\|\hat{e}\| < \|b\|$ as needed. □

Theorem B.29 (Typed Expression Expansion)**.** *If $\langle \mathcal{D}; \Delta \rangle\; \langle \mathcal{G}; \Gamma \rangle \vdash_{\hat{\Psi};\hat{\Phi}} \hat{e} \rightsquigarrow e : \tau$ then $\Delta\;\Gamma \vdash e : \tau$.*

Proof. This theorem follows immediately from Theorem B.28, part 1(a). □

B.5.4 Abstract Reasoning Principles

Lemma B.30 (Proto-Type Expansion Decomposition)**.** *If $\Delta \vdash^{\langle \mathcal{D}; \Delta_{app} \rangle; b} \grave{\tau} \rightsquigarrow \tau\;\mathsf{type}$ where $\mathsf{seg}(\grave{\tau}) = \{\mathtt{splicedt}[m_i; n_i]\}_{0 \le i < n}$ then all of the following hold:*

1. $\{\langle \mathcal{D}; \Delta_{app} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow \tau_i\;\mathsf{type}\}_{0 \le i < n}$
2. $\tau = [\{\tau_i / t_i\}_{0 \le i < n}]\tau'$ *for some* τ' *and fresh* $\{t_i\}_{0 \le i < n}$ *(i.e.* $\{t_i \notin dom(\Delta)\}_{0 \le i < n}$ *and* $\{t_i \notin dom(\Delta_{app})\}_{0 \le i < n}$*)*
3. $\mathsf{fv}(\tau') \subset dom(\Delta) \cup \{t_i\}_{0 \le i < n}$

Proof. By rule induction over Rules (B.9). In the following, let $\hat{\Delta} = \langle \mathcal{D}; \Delta_{\text{app}} \rangle$ and $\mathbb{T} = \hat{\Delta}; b$.

Case (B.9a)**.**

(1) $\grave{\tau} = t$ by assumption

(2) $\tau = t$ by assumption

(3) $\Delta = \Delta', t\ \mathsf{type}$ by assumption

(4) $\mathsf{seg}(\grave{\tau}) = \emptyset$ by definition

(5) $\mathsf{fv}(t) = \{t\}$ by definition

(6) $\{t\} \subset \mathrm{dom}(\Delta) \cup \emptyset$ by definition

The conclusions hold as follows:

1. This conclusion holds trivially because $n = 0$.
2. Choose $\tau' = t$ and $\emptyset$.
3. (6)

Case (B.9b)**.**

(1) $\grave{\tau} = \mathtt{prparr}(\grave{\tau}_1; \grave{\tau}_2)$ by assumption

(2) $\tau = \mathtt{parr}(\tau'_1; \tau'_2)$ by assumption

(3) $\Delta \vdash^{\mathbb{T}} \grave{\tau}_1 \leadsto \tau'_1\ \mathsf{type}$ by assumption

(4) $\Delta \vdash^{\mathbb{T}} \grave{\tau}_2 \leadsto \tau'_2\ \mathsf{type}$ by assumption

(5) $\mathsf{seg}(\grave{\tau}) = \mathsf{seg}(\grave{\tau}_1) \cup \mathsf{seg}(\grave{\tau}_2)$ by definition

(6) $\mathsf{seg}(\grave{\tau}_1) = \{\mathtt{splicedt}[m_i; n_i]\}_{0 \leq i < n'}$ by definition

(7) $\mathsf{seg}(\grave{\tau}_2) = \{\mathtt{splicedt}[m_i; n_i]\}_{n' \leq i < n}$ by definition

(8) $\{\langle \mathcal{D}; \Delta_{\mathrm{app}} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m_i; n_i)) \leadsto \tau_i\ \mathsf{type}\}_{0 \leq i < n'}$
by IH on (3) and (6)

(9) $\tau'_1 = [\{\tau_i / t_i\}_{0 \leq i < n'}]\tau''_1$ for some τ''_1 and fresh $\{t_i\}_{0 \leq i < n'}$
by IH on (3) and (6)

(10) $\mathsf{fv}(\tau''_1) \subset \mathrm{dom}(\Delta) \cup \{t_i\}_{0 \leq i < n'}$ by IH on (3) and (6)

(11) $\{\langle \mathcal{D}; \Delta_{\mathrm{app}} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m_i; n_i)) \leadsto \tau_i\ \mathsf{type}\}_{n' \leq i < n}$
by IH on (4) and (7)

(12) $\tau'_2 = [\{\tau_i / t_i\}_{n' \leq i < n}]\tau''_2$ for some τ''_2 and fresh $\{t_i\}_{n' \leq i < n}$
by IH on (4) and (7)

(13) $\mathsf{fv}(\tau''_2) \subset \mathrm{dom}(\Delta) \cup \{t_i\}_{n' \leq i < n}$ by IH on (4) and (7)

(14) $\{t_i\}_{0 \leq i < n'} \cap \{t_i\}_{n' \leq i < n} = \emptyset$ by identification convention

(15) $\mathsf{fv}(\tau''_1) \subset \mathrm{dom}(\Delta) \cup \{t_i\}_{0 \leq i < n}$ by (10) and (14)

(16) $\mathsf{fv}(\tau''_2) \subset \mathrm{dom}(\Delta) \cup \{t_i\}_{0 \leq i < n}$ by (13) and (14)

(17) $\tau'_1 = [\{\tau_i / t_i\}_{0 \leq i < n}]\tau''_1$ by substitution properties and (9) and (14)

(18) $\tau_2' = [\{\tau_i/t_i\}_{0\le i<n}]\tau_2''$	by substitution properties and (12) and (14)
(19) $\texttt{parr}(\tau_1';\tau_2') = [\{\tau_i/t_i\}_{0\le i<n}]\texttt{parr}(\tau_1'';\tau_2'')$	by substitution and (17) and (18)
(20) $\mathsf{fv}(\texttt{parr}(\tau_1'';\tau_2'')) = \mathsf{fv}(\tau_1'') \cup \mathsf{fv}(\tau_2'')$	by definition
(21) $\mathsf{fv}(\texttt{parr}(\tau_1'';\tau_2'')) \subset \mathrm{dom}(\Delta) \cup \{t_i\}_{0\le i<n}$	by (20) and (15) and (16)

The conclusions hold as follows:

1. (8) $\cup$ (11)
2. Choosing $\{t_i\}_{0\le i<n}$ and $\texttt{parr}(\tau_1'';\tau_2'')$, by (19)
3. (21)

Case (B.9c) **through** (B.9f)**.** These cases follow by analagous inductive argument.

Case (B.9g)**.**

(1) $\grave{\tau} = \texttt{splicedt}[m;n]$	by assumption
(2) $\mathsf{seg}(\texttt{splicedt}[m;n]) = \{\texttt{splicedt}[m;n]\}$	by definition
(3) $\mathsf{parseUTyp}(\mathsf{subseq}(b;m;n)) = \hat{\tau}$	by assumption
(4) $\langle \mathcal{D};\Delta_{\text{app}}\rangle \vdash \hat{\tau} \rightsquigarrow \tau\ \mathsf{type}$	by assumption
(5) $t \notin \mathrm{dom}(\Delta)$	by identification convention
(6) $t \notin \mathrm{dom}(\Delta_{\text{app}})$	by identification
(7) $\tau = [\tau/t]\tau$	by definition
(8) $\mathsf{fv}(t) \subset \Delta \cup \{t\}$	by definition

The conclusions hold as follows:

1. (3) and (4)
2. Choosing $\{t\}$ and t, by (5), (6) and (7)
3. (8)

$\square$

Lemma B.31 (Proto-Expression and Proto-Rule Expansion Decomposition)**.**

1. *If* $\Delta\ \Gamma \vdash^{\langle \mathcal{D};\Delta_{app}\rangle;\langle \mathcal{G};\Gamma_{app}\rangle;\hat{\Psi};\hat{\Phi};b} \grave{e} \rightsquigarrow e : \tau$ *where* $\mathsf{seg}(\grave{e}) = \{\texttt{splicedt}[m_i';n_i']\}_{0\le i<n_{ty}} \cup \{\texttt{splicede}[m_i;n_i;\grave{\tau}_i]\}_{0\le i<n_{exp}}$ *then all of the following hold:*

(a) $\{\langle \mathcal{D}; \Delta_{app}\rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \rightsquigarrow \tau'_i \text{ type}\}_{0 \le i < n_{ty}}$

(b) $\{\emptyset \vdash^{\langle \mathcal{D}; \Delta_{app}\rangle; b} \grave{\tau}_i \rightsquigarrow \tau_i \text{ type}\}_{0 \le i < n_{exp}}$

(c) $\{\langle \mathcal{D}; \Delta_{app}\rangle\ \langle \mathcal{G}; \Gamma_{app}\rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \mathsf{parseUExp}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow e_i : \tau_i\}_{0 \le i < n_{exp}}$

(d) $e = [\{\tau'_i / t_i\}_{0 \le i < n_{ty}}, \{e_i / x_i\}_{0 \le i < n_{exp}}]e'$ *for some* e' *and* $\{t_i\}_{0 \le i < n_{ty}}$ *and* $\{x_i\}_{0 \le i < n_{exp}}$ *such that* $\{t_i\}_{0 \le i < n_{ty}}$ *fresh and* $\{x_i\}_{0 \le i < n_{exp}}$ *fresh*

(e) $\mathsf{fv}(e') \subset dom(\Delta) \cup dom(\Gamma) \cup \{t_i\}_{0 \le i < n_{ty}} \cup \{x_i\}_{0 \le i < n_{exp}}$

2. *If* $\Delta\ \Gamma \vdash^{\langle \mathcal{D}; \Delta_{app}\rangle; \langle \mathcal{G}; \Gamma_{app}\rangle; \hat{\Psi}; \hat{\Phi}; b} \grave{r} \rightsquigarrow r : \tau \Rightarrow \tau'$ *and*

$$\mathsf{seg}(\grave{r}) = \{\texttt{splicedt}[m'_i; n'_i]\}_{0 \le i < n_{ty}} \cup \{\texttt{splicede}[m_i; n_i; \grave{\tau}_i]\}_{0 \le i < n_{exp}}$$

then all of the following hold:

(a) $\{\langle \mathcal{D}; \Delta_{app}\rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \rightsquigarrow \tau'_i \text{ type}\}_{0 \le i < n_{ty}}$

(b) $\{\emptyset \vdash^{\langle \mathcal{D}; \Delta_{app}\rangle; b} \grave{\tau}_i \rightsquigarrow \tau_i \text{ type}\}_{0 \le i < n_{exp}}$

(c) $\{\langle \mathcal{D}; \Delta_{app}\rangle\ \langle \mathcal{G}; \Gamma_{app}\rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \mathsf{parseUExp}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow e_i : \tau_i\}_{0 \le i < n_{exp}}$

(d) $r = [\{\tau'_i / t_i\}_{0 \le i < n_{ty}}, \{e_i / x_i\}_{0 \le i < n_{exp}}]r'$ *for some* e' *and fresh* $\{t_i\}_{0 \le i < n_{ty}}$ *and fresh* $\{x_i\}_{0 \le i < n_{exp}}$

(e) $\mathsf{fv}(r') \subset dom(\Delta) \cup dom(\Gamma) \cup \{t_i\}_{0 \le i < n_{ty}} \cup \{x_i\}_{0 \le i < n_{exp}}$

Proof. By rule induction over Rules (B.10) and Rule (B.11). In the following, let $\hat{\Delta} = \langle \mathcal{D}; \Delta_{\text{app}}\rangle$ and $\hat{\Gamma} = \langle \mathcal{G}; \Gamma_{\text{app}}\rangle$ and $\mathbb{E} = \hat{\Delta}; \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b$.

1. **Case** (B.10a).

(1) $\grave{e} = x$	by assumption
(2) $e = x$	by assumption
(3) $\Gamma = \Gamma', x : \tau$	by assumption
(4) $\mathsf{seg}(x) = \{\}$	by definition
(5) $\mathsf{fv}(x) = \{x\}$	by definition
(6) $\mathsf{fv}(x) \subset \mathrm{dom}(\Gamma)$	by definition
(7) $\mathsf{fv}(x) \subset \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta)$	by (6) and definition of subset

The conclusions hold as follows:

(a) This conclusion holds trivially because $n_{\text{ty}} = 0$.

(b) This conclusion holds trivially because $n_{\text{exp}} = 0$.

(c) This conclusion holds trivially because $n_{\text{exp}} = 0$.

(d) Choose $x, \emptyset$ and $\emptyset$.

(e) (7)

Case (B.10b) **through** (B.10m)**.** These cases follow by straightforward inductive argument.

Case (B.10n)**.**

(1) $\grave{e} = \texttt{splicede}[m; n; \grave{\tau}]$ by assumption

(2) $\mathsf{seg}(\texttt{splicede}[m; n; \grave{\tau}]) = \mathsf{seg}(\grave{\tau}) \cup \{\texttt{splicede}[m; n; \grave{\tau}]\}$ by definition

(3) $\mathsf{seg}(\grave{\tau}) = \{\texttt{splicedt}[m'_i; n'_i]\}_{0 \leq i < n_{\text{ty}}}$ by definition

(4) $\emptyset \vdash^{\mathsf{ts}(\mathbb{E})} \grave{\tau} \rightsquigarrow \tau$ type by assumption

(5) $\mathsf{parseUExp}(\mathsf{subseq}(b; m; n)) = \hat{e}$ by assumption

(6) $\langle \mathcal{D}; \Delta_{\text{app}} \rangle\ \langle \mathcal{G}; \Gamma_{\text{app}} \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{e} \rightsquigarrow e : \tau$ by assumption

(7) $\{\langle \mathcal{D}; \Delta_{\text{app}} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \rightsquigarrow \tau'_i \text{ type}\}_{0 \leq i < n_{\text{ty}}}$ by Lemma B.30 on (4) and (3)

(8) $x \notin \mathrm{dom}(\Gamma)$ by identification convention

(9) $x \notin \mathrm{dom}(\Gamma_{\text{app}})$ by identification convention

(10) $x \notin \mathrm{dom}(\Delta)$ by identificaiton convention

(11) $x \notin \mathrm{dom}(\Delta_{\text{app}})$ by identification convention

(12) $e = [\{\tau'_i / t_i\}_{0 \leq i < n_{\text{ty}}}, e/x]x$ by definition

(13) $\mathsf{fv}(x) = \{x\}$ by definition

(14) $\mathsf{fv}(x) \subset \mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma) \cup \{t_i\}_{0 \leq i < n_{\text{ty}}} \cup \{x\}$ by definition

The conclusions hold as follows:

(a) (7)

(b) $\{(4)\}$

(c) $\{(6)\}$

(d) Choosing x, $\{t_i\}_{0 \leq i < n_{\text{ty}}}$ and $\{x\}$, by (8), (9), (10), (11) and (12).

(e) (14)

Case (B.10o)**.**

(1) $\grave{e} = \texttt{prmatch}(\grave{e}'; \{\grave{r}_i\}_{1 \leq i \leq n})$ by assumption

(2) $e = \texttt{match}(\tau; e')\{r_i\}_{1 \leq i \leq n}$ by assumption

(3) $\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{e} \rightsquigarrow e : \tau'$ by assumption

(4) $\{\Delta\ \Gamma \vdash^{\mathbb{E}} \grave{r}_j \rightsquigarrow r_j : \tau' \Mapsto \tau\}_{1 \leq j \leq n}$ by assumption

(5) $\mathsf{seg}(\texttt{prmatch}(\grave{e}'; \{\grave{r}_i\}_{1 \leq i \leq n})) = \mathsf{seg}(\grave{e}) \cup \bigcup_{0 \leq i < n} \mathsf{seg}(\grave{r}_i)$ by definition

(6) $\mathsf{seg}(\grave{e}') = \{\texttt{splicedt}[m'_i; n'_i]\}_{0 \leq i < n'_{\mathsf{ty}}} \cup \{\texttt{splicede}[m_i; n_i; \grave{\tau}_i]\}_{0 \leq i < n'_{\mathsf{exp}}}$ by definition

(7) $\{\mathsf{seg}(\grave{r}_j) = \{\texttt{splicedt}[m'_{i,j}; n'_{i,j}]\}_{0 \leq i < n_{\mathsf{ty},j}} \cup \{\texttt{splicede}[m_{i,j}; n_{i,j}; \grave{\tau}_{i,j}]\}_{0 \leq i < n_{\mathsf{exp},j}}\}_{0 \leq j < n}$ by definition

(8) $\{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \leadsto \tau'_i \;\mathsf{type}\}_{0 \leq i < n'_{\mathsf{ty}}}$ by IH, part 1 on (3) and (6)

(9) $\{\emptyset \vdash^{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle; b} \grave{\tau}_i \leadsto \tau_i \;\mathsf{type}\}_{0 \leq i < n'_{\mathsf{exp}}}$ by IH, part 1 on (3) and (6)

(10) $\{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle \; \langle \mathcal{G}; \Gamma_{\mathsf{app}} \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \mathsf{parseUExp}(\mathsf{subseq}(b; m_i; n_i)) \leadsto e_i : \tau_i\}_{0 \leq i < n'_{\mathsf{exp}}}$ by IH, part 1 on (3) and (6)

(11) $e' = [\{\tau'_i / t_i\}_{0 \leq i < n'_{\mathsf{ty}}}, \{e_i / x_i\}_{0 \leq i < n'_{\mathsf{exp}}}]e''$ for some e'' and fresh $\{t_i\}_{0 \leq i < n'_{\mathsf{ty}}}$ and fresh $\{x_i\}_{0 \leq i < n'_{\mathsf{exp}}}$ by IH, part 1 on (3) and (6)

(12) $\mathsf{fv}(e'') \subset \mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma) \cup \{t_i\}_{0 \leq i < n'_{\mathsf{ty}}} \cup \{x_i\}_{0 \leq i < n'_{\mathsf{exp}}}$ by IH, part 1 on (3) and (6)

(13) $\{\{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_{i,j}; n'_{i,j})) \leadsto \tau'_{i,j} \;\mathsf{type}\}_{0 \leq i < n_{\mathsf{ty},j}}\}_{0 \leq j < n}$ by IH, part 2 over (4) and (7)

(14) $\{\{\emptyset \vdash^{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle; b} \grave{\tau}_{i,j} \leadsto \tau_{i,j} \;\mathsf{type}\}_{0 \leq i < n_{\mathsf{exp},j}}\}_{0 \leq j < n}$ by IH, part 2 over (4) and (7)

(15) $\{\{\langle \mathcal{D}; \Delta_{\mathsf{app}} \rangle \; \langle \mathcal{G}; \Gamma_{\mathsf{app}} \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \mathsf{parseUExp}(\mathsf{subseq}(b; m_{i,j}; n_{i,j})) \leadsto e_{i,j} : \tau_{i,j}\}_{0 \leq i < n_{\mathsf{exp},j}}\}_{0 \leq j < n}$ by IH, part 2 over (4) and (7)

(16) $\{r_j = [\{\tau'_{i,j} / t_{i,j}\}_{0 \leq i < n_{\mathsf{ty},j}}, \{e_{i,j} / x_{i,j}\}_{0 \leq i < n_{\mathsf{exp},j}}]r'_j\}_{0 \leq j < n}$ for some $\{r'_j\}_{0 \leq j < n}$ and fresh $\{\{t_{i,j}\}_{0 \leq i < n_{\mathsf{ty},j}}\}_{0 \leq j < n}$ and fresh $\{\{x_{i,j}\}_{0 \leq i < n_{\mathsf{exp},j}}\}_{0 \leq j < n}$ by IH, part 2 over (4) and (7)

(17) $\{\mathsf{fv}(r'_j) \subset \mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma) \cup \{t_{i,j}\}_{0 \leq i < n_{\mathsf{ty},j}} \cup \{x_{i,j}\}_{0 \leq i < n_{\mathsf{exp},j}}\}_{0 \leq j < n}$ by IH, part 2 over (4) and (7)

(18) $(\cup_{0 \leq j < n} \{t_{i,j}\}_{0 \leq i < n_{\mathsf{ty},j}}) \cap \{t_i\}_{0 \leq i < n'_{\mathsf{ty}}} = \emptyset$ by identification convention

(19) $(\cup_{0 \leq j < n} \{x_{i,j}\}_{0 \leq i < n_{\mathsf{exp},j}}) \cap \{x_i\}_{0 \leq i < n'_{\mathsf{exp}}} = \emptyset$ by identification convention

(20) $e' = [\{\tau'_i / t_i\}_{0 \leq i < n'_{\mathsf{ty}}} \cup_{0 \leq j < n} \{\tau_{i,j} / t_{i,j}\}_{0 \leq i < n_{\mathsf{ty},j}}, \{e_i / x_i\}_{0 \leq i < n_{\mathsf{exp}}} \cup_{0 \leq j < n}$

$\{\tau_{i,j}/t_{i,j}\}_{0\le i<n_{\text{ty},j}}]e''$ by substitution properties and (11) and (12) and (18) and (19)

(21) $\{r_j = [\{\tau_i'/t_i\}_{0\le i<n'_{\text{ty}}} \cup_{0\le j<n} \{\tau_{i,j}/t_{i,j}\}_{0\le i<n_{\text{ty},j}}, \{e_i/x_i\}_{0\le i<n_{\text{exp}}} \cup_{0\le j<n}$ $\{\tau_{i,j}/t_{i,j}\}_{0\le i<n_{\text{ty},j}}]r_j'\}_{0\le j<n}$ by substitution properties and (16) and (17) and (18) and (19)

(22) $e = [\{\tau_i'/t_i\}_{0\le i<n'_{\text{ty}}} \cup_{0\le j<n} \{\tau_{i,j}/t_{i,j}\}_{0\le i<n_{\text{ty},j}}, \{e_i/x_i\}_{0\le i<n'_{\text{exp}}} \cup_{0\le j<n}$ $\{e_{i,j}/x_{i,j}\}_{0\le i<n_{\text{exp},j}}]\mathtt{match}(e''; \{r_i'\}_{1\le i\le n})$ by (20) and (21) and definition of substitution

(23) $\mathsf{fv}(e'') \subset \text{dom}(\Delta) \cup \text{dom}(\Gamma) \cup \{t_i\}_{0\le i<n'_{\text{ty}}} \cup_{0\le j<n} \{t_{i,j}\}_{0\le i<n_{\text{ty},j}} \cup$ $\{x_i\}_{0\le i<n'_{\text{exp}}} \cup_{0\le j<n} \{x_{i,j}\}_{0\le i<n_{\text{exp},j}}$ by (12) and (18) and (19)

(24) $\{\mathsf{fv}(r_j') \subset \text{dom}(\Delta) \cup \text{dom}(\Gamma) \cup \{t_i\}_{0\le i<n'_{\text{ty}}} \cup_{0\le j<n} \{t_{i,j}\}_{0\le i<n_{\text{ty},j}} \cup$ $\{x_i\}_{0\le i<n'_{\text{exp}}} \cup_{0\le j<n} \{x_{i,j}\}_{0\le i<n_{\text{exp},j}}\}_{0\le j<n}$ by (17) and (18) and (19)

(25) $\mathsf{fv}(\mathtt{match}(e''; \{r_i'\}_{1\le i\le n})) \subset \text{dom}(\Delta) \cup \text{dom}(\Gamma) \cup \{t_i\}_{0\le i<n'_{\text{ty}}} \cup_{0\le j<n}$ $\{t_{i,j}\}_{0\le i<n_{\text{ty},j}} \cup \{x_i\}_{0\le i<n'_{\text{exp}}} \cup_{0\le j<n} \{x_{i,j}\}_{0\le i<n_{\text{exp},j}}$ by (23) and (24)

The conclusions hold as follows:

(a) $(8) \cup \bigcup_{0\le j<n} (13)_j$
(b) $(9) \cup \bigcup_{0\le j<n} (14)_j$
(c) $(10) \cup \bigcup_{0\le j<n} (15)_j$
(d) Choose:
  i. $\mathtt{match}(e''; \{r_i'\}_{1\le i\le n})$
  ii. $\{t_i\}_{0\le i<n'_{\text{ty}}} \cup \{\{t_{i,j}\}_{0\le i<n_{\text{ty},j}}\}_{0\le j<n}$; and
  iii. $\{x_i\}_{0\le i<n'_{\text{exp}}} \cup \{\{x_{i,j}\}_{0\le i<n_{\text{exp},j}}\}_{0\le j<n}$; and

  We have $e = [\{\tau_i'/t_i\}_{0\le i<n'_{\text{ty}}} \cup \{\{\tau_{i,j}/t_{i,j}\}_{0\le i<n_{\text{ty},j}}\}_{0\le j<n}, \{e_i/x_i\}_{0\le i<n'_{\text{exp}}} \cup \{\{e_{i,j}/x_{i,j}\}_{0\le i<n_{\text{exp},j}}\}_{0\le j<n}]\mathtt{match}(e''; \{r_i'\}_{1\le i\le n})$ by (22).
(e) (25)

2. By rule induction over the rule typing assumption. There is only one case. In the following, let $\hat{\Delta} = \langle \mathcal{D}; \Delta_{\text{app}} \rangle$ and $\hat{\Gamma} = \langle \mathcal{G}; \Gamma_{\text{app}} \rangle$ and $\mathbb{E} = \hat{\Delta}; \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b$.

Case (B.11).

(1) $\grave{r} = \texttt{prrule}(p.\grave{e})$ by assumption

(2) $r = \texttt{rule}(p.e)$ by assumption

(3) $p : \tau \dashv\!\mid \Gamma'$ by assumption

(4) $\Delta\ \Gamma \cup \Gamma' \vdash^{\mathbb{E}} \grave{e} \leadsto e : \tau'$ by assumption

(5) $\mathsf{seg}(\grave{r}) = \mathsf{seg}(\grave{e})$ by definition

(6) $\mathsf{seg}(\grave{e}) = \{\texttt{splicedt}[m'_i; n'_i]\}_{0 \le i < n_{\text{ty}}} \cup \{\texttt{splicede}[m_i; n_i; \grave{\tau}_i]\}_{0 \le i < n_{\text{exp}}}$ by definition

(7) $\{\langle \mathcal{D}; \Delta_{\text{app}} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \leadsto \tau'_i\ \mathsf{type}\}_{0 \le i < n_{\text{ty}}}$ by IH, part 1 on (4) and (6)

(8) $\{\emptyset \vdash^{\langle \mathcal{D}; \Delta_{\text{app}} \rangle; b} \grave{\tau}_i \leadsto \tau_i\ \mathsf{type}\}_{0 \le i < n_{\text{exp}}}$ by IH, part 1 on (4) and (6)

(9) $\{\langle \mathcal{D}; \Delta_{\text{app}} \rangle\ \langle \mathcal{G}; \Gamma_{\text{app}} \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \mathsf{parseUExp}(\mathsf{subseq}(b; m_i; n_i)) \leadsto e_i : \tau_i\}_{0 \le i < n_{\text{exp}}}$ by IH, part 1 on (4) and (6)

(10) $e = [\{\tau'_i / t_i\}_{0 \le i < n_{\text{ty}}}, \{e_i / x_i\}_{0 \le i < n_{\text{exp}}}]e'$ for some e' and fresh $\{t_i\}_{0 \le i < n_{\text{ty}}}$ and fresh $\{x_i\}_{0 \le i < n_{\text{exp}}}$ by IH, part 1 on (4) and (6)

(11) $\mathsf{fv}(e') \subset \mathsf{dom}(\Delta) \cup \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Gamma') \cup \{t_i\}_{0 \le i < n_{\text{ty}}} \cup \{x_i\}_{0 \le i < n_{\text{exp}}}$ by IH, part 1 on (4) and (6)

(12) $r = [\{\tau'_i / t_i\}_{0 \le i < n_{\text{ty}}}, \{e_i / x_i\}_{0 \le i < n_{\text{exp}}}]\texttt{rule}(p.e')$ by substitution properties and (10)

(13) $\mathsf{fv}(p) = \mathsf{dom}(\Gamma')$ by Lemma B.5 on (3)

(14) $\mathsf{fv}(\texttt{rule}(p.e')) \subset \mathsf{dom}(\Delta) \cup \mathsf{dom}(\Gamma) \cup \{t_i\}_{0 \le i < n_{\text{ty}}} \cup \{x_i\}_{0 \le i < n_{\text{exp}}}$ by definition of $\mathsf{fv}(r)$ and (11) and (13)

The conclusions hold as follows:

(a) (7)

(b) (8)

(c) (9)

(d) Choosing $\texttt{rule}(p.e')$ and $\{t_i\}_{0 \le i < n_{\text{ty}}}$ and $\{x_i\}_{0 \le i < n_{\text{exp}}}$, by (12)

(e) (14)

$\square$

Theorem B.32 (seTLM Abstract Reasoning Principles). *If $\langle \mathcal{D}; \Delta \rangle\ \langle \mathcal{G}; \Gamma \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{a}$ ‘(b)‘ $\leadsto e : \tau$ then:*

1. *(**Expansion Typing**) $\hat{\Psi} = \hat{\Psi}', \hat{a} \leadsto x \hookrightarrow \texttt{setlm}(\tau; e_{parse})$ and $\Delta\ \Gamma \vdash e : \tau$*

2. *(**Responsibility**) $b \downarrow_{\mathsf{Body}} e_{body}$ and $e_{parse}(e_{body}) \Downarrow \texttt{inj}[\texttt{SuccessE}](e_{proto})$ and $e_{proto} \uparrow_{\mathsf{PrExpr}} \grave{e}$*

3. *(**Segmentation**)* $\mathsf{seg}(\grave{e})$ segments b

4. *(**Segment Typing**)* $\mathsf{seg}(\grave{e}) = \{\texttt{splicedt}[m'_i; n'_i]\}_{0 \leq i < n_{ty}} \cup \{\texttt{splicede}[m_i; n_i; \grave{\tau}_i]\}_{0 \leq i < n_{exp}}$ *and*

   (a) $\{\langle \mathcal{D}; \Delta \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \rightsquigarrow \tau'_i \ \mathsf{type}\}_{0 \leq i < n_{ty}}$ *and* $\{\Delta \vdash \tau'_i \ \mathsf{type}\}_{0 \leq i < n_{ty}}$

   (b) $\{\emptyset \vdash^{\langle \mathcal{D}; \Delta \rangle; b} \grave{\tau}_i \rightsquigarrow \tau_i \ \mathsf{type}\}_{0 \leq i < n_{exp}}$ *and* $\{\Delta \vdash \tau_i \ \mathsf{type}\}_{0 \leq i < n_{exp}}$

   (c) $\{\langle \mathcal{D}; \Delta \rangle \ \langle \mathcal{G}; \Gamma \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \mathsf{parseUExp}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow e_i : \tau_i\}_{0 \leq i < n_{exp}}$ *and* $\{\Delta \ \Gamma \vdash e_i : \tau_i\}_{0 \leq i < n_{exp}}$

5. *(**Capture Avoidance**) $e = [\{\tau'_i / t_i\}_{0 \leq i < n_{ty}}, \{e_i / x_i\}_{0 \leq i < n_{exp}}]e'$ for some $\{t_i\}_{0 \leq i < n_{ty}}$ and $\{x_i\}_{0 \leq i < n_{exp}}$ and e'*

6. *(**Context Independence**)* $\mathsf{fv}(e') \subset \{t_i\}_{0 \leq i < n_{ty}} \cup \{x_i\}_{0 \leq i < n_{exp}}$

Proof. By rule induction over Rules (B.6). There is only one rule that applies. In the following, let $\hat{\Delta} = \langle \mathcal{D}; \Delta \rangle$ and $\hat{\Gamma} = \langle \mathcal{G}; \Gamma \rangle$.

Case (B.6o).

(1) $\hat{\Psi} = \hat{\Psi}', \hat{a} \rightsquigarrow x \hookrightarrow \texttt{setlm}(\tau; e_{\text{parse}})$	by assumption
(2) $\Gamma = \Gamma', x : \tau_{\text{dep}}$	by assumption
(3) $e = \texttt{ap}(e_x; x)$	by assumption
(4) $\langle \mathcal{D}; \Delta \rangle \ \langle \mathcal{G}; \Gamma \rangle \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{a} \ \texttt{`}(b)\texttt{`} \rightsquigarrow e : \tau$	by assumption
(5) $\Delta \ \Gamma \vdash e : \tau$	by Theorem B.29 on (4)
(6) $b \downarrow_{\mathsf{Body}} e_{\text{body}}$	by assumption
(7) $e_{\text{parse}}(e_{\text{body}}) \Downarrow \texttt{inj}[\texttt{SuccessE}](e_{\text{proto}})$	by assumption
(8) $e_{\text{proto}} \uparrow_{\mathsf{PrExpr}} \grave{e}$	by assumption
(9) $\mathsf{seg}(\grave{e})$ segments b	by assumption
(10) $\emptyset \ \emptyset \vdash^{\hat{\Delta}; \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b} \grave{e} \rightsquigarrow e_x : \texttt{parr}(\tau_{\text{dep}}; \tau)$	by assumption
(11) $\mathsf{seg}(\grave{e}) = \{\texttt{splicedt}[m'_i; n'_i]\}_{0 \leq i < n_{\text{ty}}} \cup \{\texttt{splicede}[m_i; n_i; \grave{\tau}_i]\}_{0 \leq i < n_{\text{exp}}}$	by definition
(12) $\{\langle \mathcal{D}; \Delta \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \rightsquigarrow \tau'_i \ \mathsf{type}\}_{0 \leq i < n_{\text{ty}}}$	by Lemma B.31 on (10) and (11)
(13) $\{\Delta \vdash \tau'_i \ \mathsf{type}\}_{0 \leq i < n_{\text{ty}}}$	by Lemma B.25, part 1 over (12)

(14) $\{\emptyset \vdash^{\langle \mathcal{D};\Delta\rangle; b} \grave{\tau}_i \leadsto \tau_i \text{ type}\}_{0\leq i<n_{\text{exp}}}$ by Lemma B.31 on (10) and (11)

(15) $\emptyset \cap \Delta = \emptyset$ by definition

(16) $\{\Delta \vdash \tau_i \text{ type}\}_{0\leq i<n_{\text{exp}}}$ by Lemma B.25, part 2 over (14) and (15)

(17) $\{\langle \mathcal{D};\Delta\rangle\ \langle \mathcal{G};\Gamma\rangle \vdash_{\hat{\Psi};\hat{\Phi}} \mathsf{parseUExp}(\mathsf{subseq}(b;m_i;n_i)) \leadsto e_i : \tau_i\}_{0\leq i<n_{\text{exp}}}$ by Lemma B.31 on (10) and (11)

(18) $\{\Delta\ \Gamma \vdash e_i : \tau_i\}_{0\leq i<n_{\text{exp}}}$ by Theorem B.29 over (17)

(19) $e_x = [\{\tau_i'/t_i\}_{0\leq i<n_{\text{ty}}}, \{e_i/x_i\}_{0\leq i<n_{\text{exp}}}]e'$ for some e' and fresh $\{t_i\}_{0\leq i<n_{\text{ty}}}$ and fresh $\{x_i\}_{0\leq i<n_{\text{exp}}}$ by Lemma B.31 on (10) and (11)

(20) $\mathsf{fv}(e') \subset \{t_i\}_{0\leq i<n_{\text{ty}}} \cup \{x_i\}_{0\leq i<n_{\text{exp}}}$ by Lemma B.31 on (10) and (11)

(21) $e = [\{\tau_i'/t_i\}_{0\leq i<n_{\text{ty}}}, \{e_i/x_i\}_{0\leq i<n_{\text{exp}}}]\mathtt{ap}(e';x)$ by definition of substitution on (19)

The conclusions hold as follows:

1. (1) and (5)
2. (6) and (7) and (8)
3. (9)
4. (11) and
   (a) (12) and (13)
   (b) (14) and (16)
   (c) (17) and (18)
5. (21)
6. (20)

□

Lemma B.33 (Proto-Pattern Expansion Decomposition). *If $\grave{p} \leadsto p : \tau \dashv\!|^{\hat{\Delta};\hat{\Phi};b}\ \hat{\Gamma}$ where*

$$\mathsf{seg}(\grave{p}) = \{\mathtt{splicedt}[m_i';n_i']\}_{0\leq i<n_{ty}} \cup \{\mathtt{splicedp}[m_i;n_i;\grave{\tau}_i]\}_{0\leq i<n_{pat}}$$

then all of the following hold:

1. $\{\hat{\Delta} \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b;m_i';n_i')) \leadsto \tau_i' \text{ type}\}_{0\leq i<n_{ty}}$
2. $\{\emptyset \vdash^{\hat{\Delta};b} \grave{\tau}_i \leadsto \tau_i \text{ type}\}_{0\leq i<n_{pat}}$

3. $\{\hat{\Delta} \vdash_{\hat{\Phi}} \mathsf{parseUPat}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow p_i : \tau_i \dashv\vert \hat{\Gamma}_i\}_{0 \leq i < n_{pat}}$

4. $\hat{\Gamma} = \biguplus_{0 \leq i < n_{pat}} \hat{\Gamma}_i$

Proof. By rule induction over Rules (B.12). In the following, let $\mathbb{P} = \hat{\Delta}; \hat{\Phi}; b$.

Case (B.12a).

(1) $\grave{p} = \mathtt{prwildp}$	by assumption
(2) $e = \mathtt{wildp}$	by assumption
(3) $\hat{\Gamma} = \langle\emptyset; \emptyset\rangle$	by assumption
(4) $\mathsf{seg}(\mathtt{prwildp}) = \emptyset$	by definition

The conclusions hold as follows:

1. This conclusion holds trivially because $n_{\mathrm{ty}} = 0$.
2. This conclusion holds trivially because $n_{\mathrm{pat}} = 0$.
3. This conclusion holds trivially because $n_{\mathrm{pat}} = 0$.
4. This conclusion holds trivially because $\hat{\Gamma} = \emptyset$ and $n_{\mathrm{pat}} = 0$.

Case (B.12b).

(1) $\grave{p} = \mathtt{prfoldp}(\grave{p}')$	by assumption
(2) $p = \mathtt{foldp}(p')$	by assumption
(3) $\tau = \mathtt{rec}(t.\tau')$	by assumption
(4) $\grave{p} \rightsquigarrow p : [\mathtt{rec}(t.\tau')/t]\tau' \dashv\vert^{\mathbb{P}} \hat{\Gamma}$	by assumption
(5) $\mathsf{seg}(\mathtt{prfoldp}(\grave{p}')) = \mathsf{seg}(\grave{p}')$	by definition
(6) $\mathsf{seg}(\grave{p}') = \{\mathtt{splicedt}[m_i'; n_i']\}_{0 \leq i < n_{\mathrm{ty}}} \cup \{\mathtt{splicedp}[m_i; n_i; \grave{\tau}_i]\}_{0 \leq i < n_{\mathrm{pat}}}$	by definition
(7) $\{\hat{\Delta} \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m_i'; n_i')) \rightsquigarrow \tau_i' \;\mathsf{type}\}_{0 \leq i < n_{\mathrm{ty}}}$	by IH on (4) and (6)
(8) $\{\emptyset \vdash^{\hat{\Delta}; b} \grave{\tau}_i \rightsquigarrow \tau_i \;\mathsf{type}\}_{0 \leq i < n_{\mathrm{pat}}}$	by IH on (4) and (6)
(9) $\{\hat{\Delta} \vdash_{\hat{\Phi}} \mathsf{parseUPat}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow p_i : \tau_i \dashv\vert \hat{\Gamma}_i\}_{0 \leq i < n_{\mathrm{pat}}}$	by IH on (4) and (6)
(10) $\hat{\Gamma} = \biguplus_{0 \leq i < n_{\mathrm{pat}}} \hat{\Gamma}_i$	by IH on (4) and (6)

The conclusions hold as follows:

1. (7)

2. (8)
3. (9)
4. (10)

Case (B.12c).

(1) $\grave{p} = \texttt{prtplp}[L](\{j \hookrightarrow \grave{p}_j\}_{j\in L})$ by assumption
(2) $p = \texttt{tplp}(\{j \hookrightarrow p_j\}_{j\in L})$ by assumption
(3) $\tau = \texttt{prod}(\{j \hookrightarrow \tau_j\}_{j\in L})$ by assumption
(4) $\hat{\Gamma} = \uplus_{j\in L} \hat{\Gamma}_j$ by assumption
(5) $\{\grave{p}_j \rightsquigarrow p_j : \tau_j \dashv\!\!|^{\mathbb{P}} \hat{\Gamma}_j\}_{j\in L}$ by assumption
(6) $\mathsf{seg}(\texttt{prtplp}[L](\{j \hookrightarrow \grave{p}_j\}_{j\in L})) = \bigcup_{j\in L} \mathsf{seg}(\grave{p}_j)$ by definition
(7) $\{\mathsf{seg}(\grave{p}_j) =$
$\{\texttt{splicedt}[m'_{i,j}; n'_{i,j}]\}_{0\leq i<n_{\text{ty},j}} \cup \{\texttt{splicedp}[m_{i,j}; n_{i,j}; \grave{\tau}_{i,j}]\}_{0\leq i<n_{\text{pat},j}}\}_{j\in L}$
by definition
(8) $n_{\text{pat}} = \Sigma_{j\in L} n_{\text{pat},j}$ by definition
(9) $\{\{\hat{\Delta} \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_{i,j}; n'_{i,j})) \rightsquigarrow \tau'_{i,j}\ \mathsf{type}\}_{0\leq i<n_{\text{ty},j}}\}_{j\in L}$
by IH over (5) and (7)
(10) $\{\{\emptyset \vdash^{\hat{\Delta}; b} \grave{\tau}_{i,j} \rightsquigarrow \tau_{i,j}\ \mathsf{type}\}_{0\leq i<n_{\text{pat},j}}\}_{j\in L}$ by IH over (5) and (7)
(11) $\{\{\hat{\Delta} \vdash_{\hat{\Phi}} \mathsf{parseUPat}(\mathsf{subseq}(b; m_{i,j}; n_{i,j})) \rightsquigarrow p_{i,j} : \tau_{i,j} \dashv\!\!| \hat{\Gamma}_{i,j}\}_{0\leq i<n_{\text{pat},j}}\}_{j\in L}$
by IH over (5) and (7)
(12) $\{\hat{\Gamma}_j = \uplus_{0\leq i<n_{\text{pat},j}} \hat{\Gamma}_{i,j}\}_{j\in L}$ by IH over (5) and (7)
(13) $\uplus_{j\in L} \hat{\Gamma}_j = \uplus_{j\in L} \uplus_{i\in n_{\text{pat},j}} \hat{\Gamma}_{i,j}$ by definition and (12)

The conclusions hold as follows:

1. $\bigcup_{j\in L} \bigcup_{i\in n_{\text{ty},j}} (9)_{i,j}$
2. $\bigcup_{j\in L} \bigcup_{i\in n_{\text{pat},j}} (10)_{i,j}$
3. $\bigcup_{j\in L} \bigcup_{i\in n_{\text{pat},j}} (11)_{i,j}$
4. (13)

Case (B.12d).

(1) $\grave{p} = \texttt{prinjp}[\ell](\grave{p}')$ by assumption
(2) $p = \texttt{injp}[\ell](p')$ by assumption
(3) $\tau = \texttt{sum}(\{i \hookrightarrow \tau_i\}_{i\in L}; \ell \hookrightarrow \tau')$ by assumption

(4) $\grave{p} \rightsquigarrow p : \tau' \dashv\!\!|^{\mathbb{P}} \hat{\Gamma}$ by assumption

(5) $\mathsf{seg}(\mathtt{prinjp}[\ell](\grave{p}')) = \mathsf{seg}(\grave{p}')$ by definition

(6) $\mathsf{seg}(\grave{p}') = \{\mathtt{splicedt}[m_i'; n_i']\}_{0 \le i < n_{\text{ty}}} \cup \{\mathtt{splicedp}[m_i; n_i; \grave{\tau}_i]\}_{0 \le i < n_{\text{pat}}}$ by definition

(7) $\{\hat{\Delta} \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m_i'; n_i')) \rightsquigarrow \tau_i' \mathsf{~type}\}_{0 \le i < n_{\text{ty}}}$ by IH on (4) and (6)

(8) $\{\emptyset \vdash^{\hat{\Delta}; b} \grave{\tau}_i \rightsquigarrow \tau_i \mathsf{~type}\}_{0 \le i < n_{\text{pat}}}$ by IH on (4) and (6)

(9) $\{\hat{\Delta} \vdash_{\hat{\Phi}} \mathsf{parseUPat}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow p_i : \tau_i \dashv\!\!| \hat{\Gamma}_i\}_{0 \le i < n_{\text{pat}}}$ by IH on (4) and (6)

(10) $\hat{\Gamma} = \biguplus_{0 \le i < n_{\text{pat}}} \hat{\Gamma}_i$ by IH on (4) and (6)

The conclusions hold as follows:

1. (7)
2. (8)
3. (9)
4. (10)

Case (B.12e)**.**

(1) $\grave{p} = \mathtt{splicedp}[m; n; \grave{\tau}]$ by assumption

(2) $\emptyset \vdash^{\hat{\Delta}; b} \grave{\tau} \rightsquigarrow \tau \mathsf{~type}$ by assumption

(3) $\mathsf{parseUPat}(\mathsf{subseq}(b; m; n)) = \hat{p}$ by assumption

(4) $\hat{\Delta} \vdash_{\hat{\Phi}} \hat{p} \rightsquigarrow p : \tau \dashv\!\!| \hat{\Gamma}$ by assumption

(5) $\mathsf{seg}(\mathtt{splicedp}[m; n; \grave{\tau}]) = \mathsf{seg}(\grave{\tau}) \cup \{\mathtt{splicedp}[m; n; \grave{\tau}]\}$ by definition

(6) $\mathsf{seg}(\grave{\tau}) = \{\mathtt{splicedt}[m_i'; n_i']\}_{0 \le i < n_{\text{ty}}}$ by definition

(7) $\{\langle \mathcal{D}; \Delta_{\text{app}} \rangle \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow \tau_i \mathsf{~type}\}_{0 \le i < n}$ by Lemma B.30 on (2) and (6)

The conclusions hold as follows:

1. (7)
2. (2)
3. (3) and (4)
4. This conclusion holds by (4) because $n_{\text{pat}} = 1$.

$\square$

Theorem B.34 (spTLM Abstract Reasoning Principles). *If $\hat{\Delta} \vdash_{\hat{\Phi}} \hat{a}$ `‘(b)‘` $\rightsquigarrow p : \tau \dashv\!\mid \hat{\Gamma}$ where $\hat{\Delta} = \langle \mathcal{D}; \Delta \rangle$ and $\hat{\Gamma} = \langle \mathcal{G}; \Gamma \rangle$ then all of the following hold:*

1. *(**Expansion Typing**) $\hat{\Phi} = \hat{\Phi}', \hat{a} \rightsquigarrow x \hookrightarrow \mathtt{sptlm}(\tau; e_{parse})$ and $p : \tau \dashv\!\mid \Gamma$*
2. *(**Responsibility**) $b \downarrow_{\mathsf{Body}} e_{body}$ and $e_{parse}(e_{body}) \Downarrow \mathtt{inj}[\mathtt{SuccessP}](e_{proto})$ and $e_{proto} \uparrow_{\mathsf{PrPat}} \grave{p}$*
3. *(**Segmentation**) $\mathsf{seg}(\grave{p})$ segments b*
4. *(**Segment Typing**) $\mathsf{seg}(\grave{p}) = \{\mathtt{splicedt}[n'_i; m'_i]\}_{0 \leq i < n_{ty}} \cup \{\mathtt{splicedp}[m_i; n_i; \grave{\tau}_i]\}_{0 \leq i < n_{pat}}$ and*
   (a) $\{\hat{\Delta} \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \rightsquigarrow \tau'_i\ \mathsf{type}\}_{0 \leq i < n_{ty}}$ and $\{\Delta \vdash \tau'_i\ \mathsf{type}\}_{0 \leq i < n_{ty}}$
   (b) $\{\emptyset \vdash^{\hat{\Delta}; b} \grave{\tau}_i \rightsquigarrow \tau_i\ \mathsf{type}\}_{0 \leq i < n_{pat}}$ and $\{\Delta \vdash \tau_i\ \mathsf{type}\}_{0 \leq i < n_{pat}}$
   (c) $\{\hat{\Delta} \vdash_{\hat{\Phi}} \mathsf{parseUPat}(\mathsf{subseq}(b; m_i; n_i)) \rightsquigarrow p_i : \tau_i \dashv\!\mid \langle \mathcal{G}_i; \Gamma_i \rangle\}_{0 \leq i < n_{pat}}$ and $\{p_i : \tau_i \dashv\!\mid \Gamma_i\}_{0 \leq i < n_{pat}}$
5. *(**Visibility**) $\mathcal{G} = \biguplus_{0 \leq i < n_{pat}} \mathcal{G}_i$ and $\Gamma = \bigcup_{0 \leq i < n_{pat}} \Gamma_i$*

Proof. By rule induction over Rules (B.8). There is only one rule that applies.

Case (B.8f).

(1)	$\hat{\Delta} \vdash_{\hat{\Phi}} \hat{a}$ `‘(b)‘` $\rightsquigarrow p : \tau \dashv\!\mid \hat{\Gamma}$	by assumption
(2)	$\hat{\Phi} = \hat{\Phi}', \hat{a} \rightsquigarrow x \hookrightarrow \mathtt{sptlm}(\tau; e_{\text{parse}})$	by assumption
(3)	$p : \tau \dashv\!\mid \Gamma$	by Theorem B.27 on (1)
(4)	$b \downarrow_{\mathsf{Body}} e_{\text{body}}$	by assumption
(5)	$e_{\text{parse}}(e_{\text{body}}) \Downarrow \mathtt{inj}[\mathtt{SuccessP}](e_{\text{proto}})$	by assumption
(6)	$e_{\text{proto}} \uparrow_{\mathsf{PrPat}} \grave{p}$	by assumption
(7)	$\mathsf{seg}(\grave{p})$ segments b	by assumption
(8)	$\grave{p} \rightsquigarrow p : \tau \dashv\!\mid^{\hat{\Delta}; \hat{\Phi}; b} \hat{\Gamma}$	by assumption
(9)	$\mathsf{seg}(\grave{p}) = \{\mathtt{splicedt}[m'_i; n'_i]\}_{0 \leq i < n_{\text{ty}}} \cup \{\mathtt{splicedp}[m_i; n_i;]\}_{0 \leq i < n_{\text{pat}}}$	by definition
(10)	$\{\hat{\Delta} \vdash \mathsf{parseUTyp}(\mathsf{subseq}(b; m'_i; n'_i)) \rightsquigarrow \tau'_i\ \mathsf{type}\}_{0 \leq i < n_{\text{ty}}}$	by Lemma B.33 on (8) and (9)
(11)	$\{\Delta \vdash \tau'_i\ \mathsf{type}\}_{0 \leq i < n_{\text{ty}}}$	by Lemma B.25, part 1 over (10)

(12) $\{\emptyset \vdash^{\hat{\Delta}; b} \grave{\tau}_i \leadsto \tau_i \ \mathsf{type}\}_{0 \leq i < n_{\text{pat}}}$ by Lemma B.33 on (8) and (9)

(13) $\{\Delta \vdash \tau_i \ \mathsf{type}\}_{0 \leq i < n_{\text{pat}}}$ by Lemma B.25, part 2 over (12)

(14) $\{\hat{\Delta} \vdash_{\hat{\Phi}} \mathsf{parseUPat}(\mathsf{subseq}(b; m_i; n_i)) \leadsto p_i : \tau_i \dashv\!\!| \ \hat{\Gamma}_i\}_{0 \leq i < n_{\text{pat}}}$ by Lemma B.33 on (8) and (9)

(15) $\{p_i : \tau_i \dashv\!\!| \ \Gamma_i\}_{0 \leq i < n_{\text{pat}}}$ by Theorem B.27 over (14)

(16) $\mathcal{G} = \biguplus_{0 \leq i < n_{\text{pat}}} \mathcal{G}_i$ and $\Gamma = \bigcup_{0 \leq i < n_{\text{pat}}} \Gamma_i$ by Lemma B.33 on (8) and (9)

The conclusions hold as follows:

1. (2) and (3)
2. (4) and (5) and (6)
3. (7)
4. (9) and
   (a) (10) and (11)
   (b) (12) and (13)
   (c) (14) and (15)
5. (16)

□

Bibliography

[1] Anton Bachin. Markup.ml — error-recovering streaming html5 and xml parsers for ocaml, 2018. http://aantron.github.io/markup.ml/. Retrieved Mar. 14, 2018. A.1c

[2] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2nd edition, 2016. URL https://www.cs.cmu.edu/~rwh/plbook/2nded.pdf. B.1