Reasonably Programmable Literal
Notation: Supplemental Material

Cyrus Omar    Jonathan Aldrich
July 8, 2018
CMU-ISR-18-104

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This report presents a complete technical account of the formal system that was described in the accompanying paper, as well as more details on the quasiquotation TLM.

This work was supported in part by AFRL and DARPA under agreement #FA8750-16-2-0042; by the NSA under lablet contract #H98230-14-C-0140.
Keywords: extensible syntax; macros; hygiene; type systems
## Contents

A Implementing Quasiquotation ........................................... 2

B ML\textsuperscript{Lit}: A Calculus of Simple TLMs .......................... 3
   B.1 Typographic Conventions ............................................ 4
   B.2 Core Language .......................................................... 5
      B.2.1 Syntax ............................................................... 5
      B.2.2 Static Semantics ................................................ 5
      B.2.3 Structural Operational Semantics .............................. 11
   B.3 Unexpanded Language (UL) ........................................... 12
      B.3.1 Syntax ............................................................... 12
      B.3.2 Type Expansion ................................................... 15
      B.3.3 Typed Expression Expansion .................................... 16
   B.4 Proto-Expansion Validation .......................................... 21
      B.4.1 Syntax of Proto-Expansions ..................................... 21
      B.4.2 Proto-Type Validation .......................................... 26
      B.4.3 Proto-Expression Validation ................................. 26
      B.4.4 Proto-Pattern Validation ....................................... 28
   B.5 Metatheory ............................................................... 28
      B.5.1 Type Expansion ................................................... 28
      B.5.2 Typed Pattern Expansion ....................................... 31
      B.5.3 Typed Expression Expansion .................................... 34
      B.5.4 Abstract Reasoning Principles ............................... 41
Appendix A

Implementing Quasiquote

The Reason quasiquotation TLMs, e.g. $\text{proto}_{\text{expr}}$, can 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, $\text{re}_{\text{expr}}$ 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.

```plaintext
// 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"])))
```
Appendix B

\textbf{ML}^{\text{Lit}}: A Calculus of Simple TLMs

This section defines \textbf{ML}^{\text{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 \textbf{ML}^{\text{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

Our typographic conventions below are based closely on PFPL’s typographic conventions for abstract binding trees [1]. 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 are assumed to be distinct.

We write $\{i \mapsto \tau_i\}_{i \in L}$ for an unordered collection of type arguments $\tau_i$, one for each $i \in L$, and similarly for arguments of other sorts. Similarly, we write $\{i \mapsto J_i\}_{i \in L}$ 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 judgments, and non-empty finite sets and finite functions are written as comma-separated sequences identified implicitly up to exchange and contraction.
B.2 Core Language

B.2.1 Syntax

<table>
<thead>
<tr>
<th>Sort</th>
<th>Operational Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typ</td>
<td>τ ::= t</td>
<td>variable</td>
</tr>
<tr>
<td></td>
<td>parr(τ; τ)</td>
<td>partial function</td>
</tr>
<tr>
<td></td>
<td>all(t.τ)</td>
<td>polymorphic</td>
</tr>
<tr>
<td></td>
<td>rec(t.τ)</td>
<td>recursive</td>
</tr>
<tr>
<td></td>
<td>prod({i ↦ τ_i}_{i∈L})</td>
<td>labeled product</td>
</tr>
<tr>
<td></td>
<td>sum({i ↦ τ_i}_{i∈L})</td>
<td>labeled sum</td>
</tr>
</tbody>
</table>

| Exp  | e ::= x           | variable |
|      | lam{τ}(x.e)       | abstraction |
|      | ap(e; e)          | application |
|      | tlam(t.e)         | type abstraction |
|      | tap{τ}(e)         | type application |
|      | fold(e)           | fold |
|      | unfold(e)         | unfold |
|      | tpl({i ↦ e_i}_{i∈L}) | labeled tuple |
|      | prj[ℓ](e)         | projection |
|      | inj[ℓ](e)         | injection |
|      | case(e; {i ↦ x_i.e_i}_{i∈L}) | case analysis |

| Rule | r ::= rule(p.e)  | rule |
| Pat  | p ::= x          | variable pattern |
|      | wildp            | wildcard pattern |
|      | foldp(p)         | fold pattern |
|      | tlp{p_i}_{i∈L}   | labeled tuple pattern |
|      | injp[ℓ](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  Δ for Δ extended with the hypothesis t type.

Typing contexts, Γ, are finite functions that map each variable x ∈ dom(Γ), where dom(Γ) is a finite set of variables, to the hypothesis x : τ, for some τ. We write Γ, x : τ, when x /∈ dom(Γ), for the extension of Γ with a mapping from x to x : τ, and Γ ∪ Γ′ when dom(Γ) ∩ dom(Γ′) = ∅ for the typing context mapping each x ∈ dom(Γ) ∪ dom(Γ′) to x : τ if x : τ ∈ Γ or x : τ ∈ Γ′. We write Δ ⊢ Γ ctx if every type in Γ is well-formed relative to Δ.

Definition B.2.1 (Typing Context Formation). Δ ⊢ Γ ctx iff for each hypothesis x : τ ∈ Γ, we have Δ ⊢ τ type.
\[ \Delta \vdash \tau \text{ type} \quad \text{\( \tau \) is a well-formed type} \]

\[ \frac{\Delta, \tau \text{ type} \vdash \tau \text{ type}}{\Delta, t \text{ type} \vdash t \text{ type}} \quad \text{(B.1a)} \]

\[ \frac{\Delta \vdash \tau_1 \text{ type} \quad \Delta \vdash \tau_2 \text{ type}}{\Delta \vdash \text{parr}(\tau_1; \tau_2) \text{ type}} \quad \text{(B.1b)} \]

\[ \frac{\Delta, t \text{ type} \vdash \tau \text{ type}}{\Delta \vdash \text{all}(t, \tau) \text{ type}} \quad \text{(B.1c)} \]

\[ \frac{\Delta, t \text{ type} \vdash \tau \text{ type}}{\Delta \vdash \text{rec}(t, \tau) \text{ type}} \quad \text{(B.1d)} \]

\[ \frac{\{\Delta \vdash \tau_i \text{ type}\}_{i \in L}}{\Delta \vdash \text{prod}\{i \mapsto \tau_i\}_{i \in L} \text{ type}} \quad \text{(B.1e)} \]

\[ \frac{\{\Delta \vdash \tau_i \text{ type}\}_{i \in L}}{\Delta \vdash \text{sum}\{i \mapsto \tau_i\}_{i \in L} \text{ type}} \quad \text{(B.1f)} \]

\[ \Delta \vdash e : \tau \quad \text{\( e \) is assigned type \( \tau \)} \]

\[ \frac{\Delta, x : \tau \vdash x : \tau}{\Delta, x : \tau \vdash x : \tau} \quad \text{(B.2a)} \]

\[ \frac{\Delta \vdash \tau \text{ type} \quad \Delta, x : \tau \vdash e : \tau'}{\Delta, x : \tau \vdash \text{lam}\{\tau\}(x, e) : \text{parr}(\tau; \tau')} \quad \text{(B.2b)} \]

\[ \frac{\Delta \vdash e_1 : \text{parr}(\tau; \tau') \quad \Delta \vdash e_2 : \tau}{\Delta \vdash \text{ap}(e_1; e_2) : \tau'} \quad \text{(B.2c)} \]

\[ \frac{\Delta, t \text{ type} \vdash e : \tau}{\Delta \vdash \text{tlam}(t, e) : \text{all}(t, \tau)} \quad \text{(B.2d)} \]

\[ \frac{\Delta \vdash e : \text{all}(t, \tau) \quad \Delta \vdash \tau' \text{ type}}{\Delta \vdash \text{tap}\{\tau'\}(e) : [\tau' / t] \tau} \quad \text{(B.2e)} \]

\[ \frac{\Delta \vdash e : [\text{rec}(t, \tau) / t] \tau}{\Delta \vdash \text{fold}(e) : \text{rec}(t, \tau)} \quad \text{(B.2f)} \]

\[ \frac{\Delta \vdash e : \text{rec}(t, \tau)}{\Delta \vdash \text{unfold}(e) : [\text{rec}(t, \tau) / t] \tau} \quad \text{(B.2g)} \]

\[ \frac{\{\Delta \vdash e_i : \tau_i\}_{i \in L}}{\Delta \vdash \text{tpl}\{i \mapsto e_i\}_{i \in L} : \text{prod}\{i \mapsto \tau_i\}_{i \in L}} \quad \text{(B.2h)} \]
\[
\Delta \Gamma \vdash e : \text{prod}(\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau) \\
\Delta \Gamma \vdash \text{prj}[\ell](e) : \tau \\
\Delta \Gamma \vdash e : \tau
\]

(B.2i)

\[
\Delta \Gamma \vdash \text{inj}[\ell](e) : \text{sum}(\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau)
\]

(B.2j)

\[
\Delta \Gamma \vdash e : \text{sum}(\{i \mapsto \tau_i\}_{i \in L}) \\
\quad \{\Delta \Gamma, x_i : \tau_i \vdash e_i : \tau\}_{i \in L}
\]

\[
\Delta \Gamma \vdash \text{case}(e; \{i \mapsto x_i.e_i\}_{i \in L}) : \tau
\]

(B.2k)

\[
\Delta \Gamma \vdash e : \tau \quad \{\Delta \Gamma \vdash r_i : \tau \Rightarrow \tau'\}_{1 \leq i \leq n}
\]

\[
\Delta \Gamma \vdash \text{match}(e; \{r_i\}_{1 \leq i \leq n}) : \tau'
\]

(B.2l)

\[
\Delta \Gamma \vdash r : \tau \Rightarrow \tau'
\]

\[
p : \tau \vdash \Gamma' \quad \Delta \Gamma \cup \Gamma' \vdash e : \tau'
\]

\[
\Delta \Gamma \vdash \text{rule}(p.e) : \tau \Rightarrow \tau'
\]

(B.3)

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

\[
p : \tau \vdash \Gamma
\]

\[
p \text{ matches values of type } \tau \text{ and generates hypotheses } \Gamma
\]

\[
\Delta \Gamma \vdash x : \tau \vdash x : \tau
\]

(B.4a)

\[
\text{wildp} : \tau \vdash \emptyset
\]

(B.4b)

\[
p : \text{rec}(t.\tau)/t \vdash \Gamma
\]

\[
\text{foldp}(p) : \text{rec}(t.\tau) \vdash \Gamma
\]

(B.4c)

\[
\{p_i : \tau_i \vdash \Gamma_i\}_{i \in L}
\]

\[
\text{tplp}(\{i \mapsto p_i\}_{i \in L}) : \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \vdash \bigcup_{i \in L} \Gamma_i
\]

(B.4d)

\[
p : \tau \vdash \Gamma
\]

\[
\text{inj}[\ell](p) : \text{sum}(\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau) \vdash \Gamma
\]

(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.2.2 (Weakening).

1. If $\Delta \vdash \tau$ type then $\Delta, t$ type $\vdash \tau$ type.

2. (a) If $\Delta \Gamma \vdash e : \tau$ then $\Delta, t$ type $\Gamma \vdash e : \tau$.
   
   (b) If $\Delta \Gamma \vdash r : \tau \Rightarrow \tau'$ then $\Delta, t$ type $\Gamma \vdash r : \tau \Rightarrow \tau'$.

3. (a) If $\Delta \Gamma \vdash e : \tau$ and $\Delta \vdash \tau''$ type then $\Delta \Gamma, x : \tau'' \vdash e : \tau$.
   
   (b) If $\Delta \Gamma \vdash r : \tau \Rightarrow \tau'$ and $\Delta \vdash \tau''$ type then $\Delta \Gamma, x : \tau'' \vdash r : \tau \Rightarrow \tau'$.

4. If $p : \tau \models \Gamma$ then $\Delta, t$ type $\vdash p : \tau \models \Gamma$.

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.


Note clause 4, which allows weakening of $\Delta$ 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.2.3 (Substitution).

1. If $\Delta, t$ type $\vdash \tau$ type and $\Delta \vdash \tau'$ type then $\Delta \vdash [\tau'/t] \tau$ type.

2. (a) If $\Delta, t$ type $\Gamma \vdash e : \tau$ and $\Delta \vdash \tau'$ type then $\Delta [\tau'/t] \Gamma \vdash [\tau'/t] e : [\tau'/t] \tau$.
   
   (b) If $\Delta, t$ type $\Gamma \vdash r : \tau \Rightarrow \tau''$ and $\Delta \vdash \tau'$ type then $\Delta [\tau'/t] \Gamma \vdash [\tau'/t] r : [\tau'/t] \tau \Rightarrow [\tau'/t] \tau''$.

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 $\Delta \Gamma, x : \tau' \vdash r : \tau \Rightarrow \tau''$ and $\Delta \Gamma \vdash e' : \tau''$ then $\Delta \Gamma \vdash [e'/x] r : \tau \Rightarrow \tau''$.

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.2.4 (Decomposition).

1. If $\Delta \vdash [\tau'/t]\tau$ type and $\Delta, t \vdash \tau$ type then $\Delta, t \vdash \tau$ type.

2. (a) If $\Delta \vdash [\tau'/t]\Gamma \vdash [\tau'/t]e : [\tau'/t]\tau$ and $\Delta \vdash \tau$ type then $\Delta, t \vdash e : \tau$.
   
   (b) If $\Delta \vdash [\tau'/t]\Gamma \vdash [\tau'/t]r : [\tau'/t]\tau \Rightarrow [\tau'/t]\tau''$ and $\Delta \vdash \tau$ type then $\Delta, t \vdash r : \tau \Rightarrow \tau''$.

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 $\tau'$.

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 $\Delta \vdash [\tau'/t]\Gamma \vdash [\tau'/t]e : [\tau'/t]\tau$ or $\Delta \vdash [\tau'/t]\Gamma \vdash [\tau'/t]r : [\tau'/t]\tau \Rightarrow [\tau'/t]\tau''$ does not depend on the form of $\tau'$.

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 $\Delta \vdash [e'/x]e : \tau$ or $\Delta \vdash [e'/x]r : \tau \Rightarrow \tau''$ does not depend on the form of $e'$.

Lemma B.2.5 (Pattern Regularity). If $p : \tau \parallel \Gamma$ and $\Delta \vdash \tau$ type then $\Delta \vdash \Gamma$ ctx and $\text{patvars}(p) = \text{dom}(\Gamma)$.

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.2.1 on (3)
(5) $\text{fv}(p) = \text{dom}(\Gamma) = \{x\}$ by definition

Case (B.4b).
(1) \( p = \text{wildp} \) by assumption
(2) \( \Gamma = \emptyset \) by assumption
(3) \( \Delta \vdash \emptyset \text{ ctx} \) by Definition B.2.1
(4) \( \text{patvars}(p) = \text{dom}(\Gamma) = \emptyset \) by definition

Case (B.4d).

(1) \( p = \text{tplp}\{i \mapsto p_i\}_{i \in L} \) by assumption
(2) \( \tau = \text{prod}\{i \mapsto \tau_i\}_{i \in L} \) by assumption
(3) \( \Gamma = \bigcup_{i \in L} \Gamma_i \) by assumption
(4) \( \{p_i : \tau_i \vdash \Gamma_i\}_{i \in L} \) by assumption
(5) \( \Delta \vdash \text{prod}\{i \mapsto \tau_i\}_{i \in L} \text{ type} \) by assumption
(6) \( \{\Delta \vdash \tau_i \text{ type}\}_{i \in L} \) by Inversion of Rule (B.1e) on (5)
(7) \( \{\Delta \vdash \Gamma_i \text{ ctx}\}_{i \in L} \) by IH over (4) and (6)
(8) \( \{\text{patvars}(p_i) = \text{dom}(\Gamma_i)\}_{i \in L} \) by IH over (4) and (6)
(9) \( \Delta \vdash \bigcup_{i \in L} \Gamma_i \text{ ctx} \) by Definition B.2.1 over (7), then Definition B.2.1 iteratively
(10) \( \text{patvars}(p) = \text{dom}(\Gamma) = \emptyset \) by definition and (8)

Case (B.4e).

(1) \( p = \text{injp}[\ell](p') \) by assumption
(2) \( \tau = \text{sum}\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau' \) by assumption
(3) \( \Delta \vdash \text{sum}\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau' \text{ type} \) by assumption
(4) \( p' : \tau' \vdash \Gamma \) by assumption
(5) \( \Delta \vdash \tau' \text{ type} \) by Inversion of Rule (B.1f) on (3)
(6) \( \Delta \vdash \Gamma \text{ ctx} \) by IH on (4) and (5)
(7) \( \text{patvars}(p') = \text{dom}(\Gamma) \) by IH on (4) and (5)
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:

<table>
<thead>
<tr>
<th>Judgement Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e \mapsto e' )</td>
<td>( e ) transitions to ( e' )</td>
</tr>
<tr>
<td>( e ) \text{ val}</td>
<td>( e ) is a value</td>
</tr>
<tr>
<td>( e ) \text{ matchfail}</td>
<td>( e ) raises match failure</td>
</tr>
</tbody>
</table>

We also define auxiliary judgements for iterated transition, \( e \mapsto \ast e' \), and evaluation, \( e \Downarrow e' \).

**Definition B.2.6 (Iterated Transition).** Iterated transition, \( e \mapsto \ast e' \), is the reflexive, transitive closure of the transition judgement, \( e \mapsto e' \).

**Definition B.2.7 (Evaluation).** \( e \Downarrow e' \) iff \( e \mapsto \ast e' \) and \( e' \) \text{ 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.2.8 (Canonical Forms).** If \( \vdash e : \tau \) and \( e \) \text{ val} then:

1. If \( \tau = \text{parr}(\tau_1; \tau_2) \) then \( e = \text{lam}(\tau_1)(x.e') \) and \( x : \tau_1 \vdash e' : \tau_2 \).
2. If \( \tau = \text{all}(t.\tau') \) then \( e = \text{tlam}(t.e') \) and \( t \text{ type} \vdash e' : \tau' \).
3. If \( \tau = \text{rec}(t.\tau') \) then \( e = \text{fold}(e') \) and \( \vdash e' : [\text{rec}(t.\tau')/t] \tau' \) and \( e' \) \text{ val}.
4. If \( \tau = \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \) then \( e = \text{tpl}(\{i \mapsto e_i\}_{i \in L}) \) and \( \vdash e_i : \tau_i \) and \( e_i \) \text{ val} for each \( i \in L \).
5. If \( \tau = \text{sum}(\{i \mapsto \tau_i\}_{i \in L'}) \) then for some label set \( L' \) and label \( \ell \) and type \( \tau' \), we have that \( \ell = L', \ell \) and \( \tau = \text{sum}(\{i \mapsto \tau_i\}_{i \in L'}; \ell \mapsto \tau') \) and \( e = \text{inj}[\ell](e') \) and \( \vdash e' : \tau' \) and \( e' \) \text{ val}.

**Condition B.2.9 (Preservation).** If \( \vdash e : \tau \) and \( e \mapsto e' \) then \( \vdash e' : \tau \).

**Condition B.2.10 (Progress).** If \( \vdash e : \tau \) then either \( e \) \text{ val} or \( e \) \text{ matchfail} or there exists an \( e' \) such that \( e \mapsto e' \).
### B.3 Unexpanded Language (UL)

#### B.3.1 Syntax

**Stylized Syntax**

<table>
<thead>
<tr>
<th>Sort</th>
<th>Stylized Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTyp</td>
<td>( \hat{\tau} ::= \hat{\iota} )</td>
<td>identifier</td>
</tr>
<tr>
<td></td>
<td>( \hat{\iota} \rightarrow \hat{\iota} )</td>
<td>partial function</td>
</tr>
<tr>
<td></td>
<td>( \forall \hat{\iota}. \hat{\tau} )</td>
<td>polymorphic</td>
</tr>
<tr>
<td></td>
<td>( \mu \hat{\iota}. \hat{\tau} )</td>
<td>recursive</td>
</tr>
<tr>
<td></td>
<td>( \langle { i \mapsto \hat{\tau}<em>i } }</em>{i \in L} )</td>
<td>labeled product</td>
</tr>
<tr>
<td></td>
<td>( { i \mapsto \hat{\tau}<em>i } }</em>{i \in L} )</td>
<td>labeled sum</td>
</tr>
<tr>
<td>UExp</td>
<td>( \hat{e} ::= \hat{x} )</td>
<td>identifier</td>
</tr>
<tr>
<td></td>
<td>( \hat{e} : \hat{\tau} )</td>
<td>ascription</td>
</tr>
<tr>
<td></td>
<td>let val ( \hat{x} = \hat{e} ) in ( \hat{e} )</td>
<td>value binding</td>
</tr>
<tr>
<td></td>
<td>( \lambda \hat{x} : \hat{\tau}. \hat{e} )</td>
<td>abstraction</td>
</tr>
<tr>
<td></td>
<td>( \hat{e}(\hat{e}) )</td>
<td>application</td>
</tr>
<tr>
<td></td>
<td>( \Lambda \hat{\iota}. \hat{e} )</td>
<td>type abstraction</td>
</tr>
<tr>
<td></td>
<td>( \hat{e}[\hat{\tau}] )</td>
<td>type application</td>
</tr>
<tr>
<td></td>
<td>fold(( \hat{e} ))</td>
<td>fold</td>
</tr>
<tr>
<td></td>
<td>unfold(( \hat{e} ))</td>
<td>unfold</td>
</tr>
<tr>
<td></td>
<td>( \langle { i \mapsto \hat{e}<em>i } }</em>{i \in L} )</td>
<td>labeled tuple</td>
</tr>
<tr>
<td></td>
<td>( \hat{e} : \ell )</td>
<td>projection</td>
</tr>
<tr>
<td></td>
<td>inj[( \ell )](( \hat{e} ))</td>
<td>injection</td>
</tr>
<tr>
<td></td>
<td>case ( \hat{e} ) ( { i \mapsto \hat{x}_i, \hat{e}_i } ) ( i \in L )</td>
<td>case analysis</td>
</tr>
<tr>
<td></td>
<td>notation ( \hat{a} ) at ( \hat{\tau} )</td>
<td>seTLM definition</td>
</tr>
<tr>
<td></td>
<td>{ expr parser ( e ); expansions require ( \hat{e} ) } in ( \hat{e} ) ( \hat{a} ) ( \cdot ) ( (b) ) }</td>
<td>seTLM application</td>
</tr>
<tr>
<td>URule</td>
<td>( \hat{\rho} ::= \hat{\rho} \Rightarrow \hat{e} )</td>
<td>match</td>
</tr>
<tr>
<td>UPat</td>
<td>( \hat{\rho} ::= \hat{x} )</td>
<td>identifier pattern</td>
</tr>
<tr>
<td></td>
<td>( \hat{\rho} )</td>
<td>wildcard pattern</td>
</tr>
<tr>
<td></td>
<td>fold(( \hat{\rho} ))</td>
<td>fold pattern</td>
</tr>
<tr>
<td></td>
<td>( \langle { i \mapsto \hat{\rho}<em>i } }</em>{i \in L} )</td>
<td>labeled tuple pattern</td>
</tr>
<tr>
<td></td>
<td>inj[( \ell )](( \hat{\rho} ))</td>
<td>injection pattern</td>
</tr>
<tr>
<td></td>
<td>( \hat{a} ) ( \cdot ) ( (b) ) }</td>
<td>spTLM application</td>
</tr>
</tbody>
</table>
**Body Lengths**  We write $\| b \|$ for the length of $b$. The metafunction $\| e \|$ computes the sum of the lengths of expression literal bodies in $e$:

$$
\begin{align*}
\| \hat{x} \| &= 0 \\
\| \hat{e} : \hat{t} \| &= \| \hat{e} \| \\
\| \text{let val } \hat{x} = \hat{e}_1 \text{ in } \hat{e}_2 \| &= \| \hat{e}_1 \| + \| \hat{e}_2 \| \\
\| \lambda \hat{x} : \hat{t}. \hat{e} \| &= \| \hat{e} \| \\
\| \hat{e}_1 (\hat{e}_2) \| &= \| \hat{e}_1 \| + \| \hat{e}_2 \| \\
\| \Lambda \hat{e} \| &= \| \hat{e} \| \\
\| \hat{e}[\hat{t}] \| &= \| \hat{e} \| \\
\| \text{fold}(\hat{e}) \| &= \| \hat{e} \| \\
\| \text{unfold}(\hat{e}) \| &= \| \hat{e} \| \\
\| \{ i \mapsto \hat{e}_i \}_{i \in L} \| &= \sum_{i \in L} \| \hat{e}_i \| \\
\| \ell : \hat{e} \| &= \| \hat{e} \| \\
\| \text{inj}[\ell](\hat{e}) \| &= \| \hat{e} \| \\
\| \text{case } \hat{e} \{ i \mapsto \hat{e}_i \}_{i \in L} \| &= \| \hat{e} \| + \sum_{i \in L} \| \hat{e}_i \| \\
\| \text{notation } \hat{a} \text{ at } \hat{t} \{ \text{ expr parser } e; \text{ expansions require } \hat{e} \} \text{ in } \hat{e}' \| &= \| \hat{e} \| + \| \hat{e}' \| \\
\| \hat{e} \{ i \mapsto \hat{e}_i \}_{i \leq n} \| &= \| \hat{e} \| + \sum_{i \leq n} \| r_i \| \\
\| \text{notation } \hat{a} \text{ at } \hat{t} \{ \text{ pat parser } e \} \text{ in } \hat{e} \| &= \| \hat{e} \|
\end{align*}
$$

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

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

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

$$
\begin{align*}
\| \hat{x} \| &= 0 \\
\| \text{fold}(\hat{p}) \| &= \| \hat{p} \| \\
\| \{ i \mapsto \hat{p}_i \}_{i \in L} \| &= \sum_{i \in L} \| \hat{p}_i \| \\
\| \text{inj}[\ell](\hat{p}) \| &= \| \hat{p} \| \\
\| \hat{a} \{ (b) \} \| &= \| b \|
\end{align*}
$$

**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 $\hat{t}$.  

13
• Each type, $\tau$, maps onto an unexpanded type, $U(\tau)$, as follows:

\[
U(t) = \hat{t} \\
U(\text{parr}(\tau_1; \tau_2)) = U(\tau_1) \rightarrow U(\tau_2) \\
U(\text{all}(t; \tau)) = \forall t . U(\tau) \\
U(\text{rec}(t; \tau)) = \mu \hat{t} . U(\tau) \\
U(\text{prod}([i \mapsto \tau_i]_{i \in L})) = \{ [i \mapsto U(\tau_i)]_{i \in L} \\
U(\text{sum}([i \mapsto \tau_i]_{i \in L})) = \{ [i \mapsto U(\tau_i)]_{i \in L} \\
\]

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

• Each core language expression, $e$, maps onto an unexpanded expression, $U(e)$, as follows:

\[
U(x) = \hat{x} \\
U(\lambda \tau . e) = \lambda \hat{x} . U(\tau).U(e) \\
U(e_1 . e_2) = U(e_1)(U(e_2)) \\
U(\tau) = \hat{\lambda} . U(e) \\
U(\text{tap}(\tau)(e)) = U(e)[U(\tau)] \\
U(\text{fold}(e)) = \text{fold}(U(e)) \\
U(\text{unfold}(e)) = \text{unfold}(U(e)) \\
U(\text{tpl}([i \mapsto e_i]_{i \in L})) = \{ [i \mapsto U(e_i)]_{i \in L} \\
U(\text{prj}[\ell](e)) = U(e) \cdot \ell \\
U(\text{inj}[\ell](e)) = \text{inj}[\ell](U(e)) \\
U(\text{match}(e; \{r_i\}_{1 \leq i \leq n})) = \text{match}(U(e) \{U(r_i)\}_{1 \leq i \leq n})
\]

• Each core language rule, $r$, maps onto an unexpanded rule, $U(r)$, as follows:

\[
U(\text{rule}(p; e)) = \text{urule}(U(p).U(e))
\]

• Each core language pattern, $p$, maps onto the unexpanded pattern, $U(p)$, as follows:

\[
U(x) = \hat{x} \\
U(\text{wildp}) = \text{uwildp} \\
U(\text{foldp}(p)) = \text{ufoldp}(U(p)) \\
U(\text{tplp}([i \mapsto p_i]_{i \in L})) = \text{utplp}[L]([i \mapsto U(p_i)]_{i \in L}) \\
U(\text{injp}[\ell](p)) = \text{uinjp}[\ell](U(p))
\]

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 $\text{parseUTyp}(b)$ and $\text{parseUExp}(b)$ and $\text{parseUPat}(b)$.

Condition B.3.1 (Textual Representability).
1. For each $\hat{\tau}$, there exists $b$ such that $\text{parseUTyp}(b) = \hat{\tau}$.

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

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

We also impose the following technical conditions:

**Condition B.3.2** (Expression Parsing Monotonicity). If $\text{parseUExp}(b) = \hat{e}$ then $\|\hat{e}\| < \|b\|$.

**Condition B.3.3** (Pattern Parsing Monotonicity). If $\text{parseUPat}(b) = \hat{p}$ then $\|\hat{p}\| < \|b\|$.

### B.3.2 Type Expansion

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

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

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

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

**Definition B.3.4** (Unexpanded Type Formation Context Formation). $\vdash \langle D; \Delta \rangle$ utctx iff for each $\hat{t} \rightsquigarrow t$ type $\in D$ we have $t$ type $\in \Delta$.

$$\hat{\Delta} \vdash \hat{t} \rightsquigarrow \tau \text{ type} \quad \hat{t} \text{ has well-formed expansion } \tau$$

$$\hat{\Delta}, \hat{t} \rightsquigarrow t \text{ type} \vdash \hat{t} \rightsquigarrow t \text{ type}$$

$$\hat{\Delta} \vdash \hat{t}_1 \rightsquigarrow \tau_1 \text{ type} \quad \hat{\Delta} \vdash \hat{t}_2 \rightsquigarrow \tau_2 \text{ type}$$

$$\hat{\Delta} \vdash \text{uparr}(\hat{t}_1; \hat{t}_2) \rightsquigarrow \text{parr}(\tau_1; \tau_2) \text{ type}$$

$$\hat{\Delta}, \hat{t} \rightsquigarrow t \text{ type} \vdash \hat{t} \rightsquigarrow t \text{ type}$$

$$\hat{\Delta} \vdash \text{uall}(\hat{t}; \hat{\tau}) \rightsquigarrow \text{all}(t.\tau) \text{ type}$$

$$\hat{\Delta}, \hat{t} \rightsquigarrow t \text{ type} \vdash \hat{t} \rightsquigarrow t \text{ type}$$

$$\hat{\Delta} \vdash \text{urec}(\hat{t}; \hat{\tau}) \rightsquigarrow \text{rec}(t.\tau) \text{ type}$$

$$\{\hat{\Delta} \vdash \hat{t}_i \rightsquigarrow \tau_i \text{ type}\}_{i \in L}$$

$$\hat{\Delta} \vdash \text{uprod}[L](\{i \rightsquigarrow \hat{t}_i\}_{i \in L}) \rightsquigarrow \text{prod}(\{i \rightsquigarrow \tau_i\}_{i \in L}) \text{ type}$$

$$\{\hat{\Delta} \vdash \hat{t}_i \rightsquigarrow \tau_i \text{ type}\}_{i \in L}$$

$$\hat{\Delta} \vdash \text{usum}[L](\{i \rightsquigarrow \hat{t}_i\}_{i \in L}) \rightsquigarrow \text{sum}(\{i \rightsquigarrow \tau_i\}_{i \in L}) \text{ type}$$

(B.5a)  

(B.5b)  

(B.5c)  

(B.5d)  

(B.5e)  

(B.5f)
B.3.3 Typed Expression Expansion

Contexts

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

We define \( \hat{\Gamma}, \hat{x} \leadsto x : \tau \) when \( \hat{\Gamma} = \langle G; \Gamma \rangle \) as an abbreviation of \( \langle G \cup \hat{x} \leadsto x; \Gamma, x : \tau \rangle \)

**Definition B.3.5 (Unexpanded Typing Context Formation).** \( \Delta \vdash \langle G; \Gamma \rangle \) uctx iff \( \Delta \vdash \Gamma \) ctx and for each \( \hat{x} \leadsto x \in G \), we have \( x \in \text{dom}(\Gamma) \).

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 \( b \downarrow_{\text{Body}} e_{\text{body}} \). An inverse mapping is defined by the body decoding judgement \( e_{\text{body}} \uparrow_{\text{Body}} b \).

<table>
<thead>
<tr>
<th>Judgement Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b \downarrow_{\text{Body}} e )</td>
<td>( b ) has encoding ( e )</td>
</tr>
<tr>
<td>( e \uparrow_{\text{Body}} b )</td>
<td>( e ) has decoding ( b )</td>
</tr>
</tbody>
</table>

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

**Condition B.3.6 (Body Isomorphism).**

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

We also assume a partial metafunction, \( \text{subseq}(b; m; n) \), 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.3.7 (Body Subsequencing).** If \( \text{subseq}(b; m; n) = b' \) then \( ||b'|| \leq ||b|| \).
Parse Results

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

\[
L_{SE} \overset{\text{def}}{=} \text{Error, SuccessE}
\]

\[
\text{ParseResultE} \overset{\text{def}}{=} \text{sum} (\text{Error} \rightarrow \langle \rangle, \text{SuccessE} \rightarrow \text{PrExpr})
\]

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

\[
L_{SP} \overset{\text{def}}{=} \text{Error, SuccessP}
\]

\[
\text{ParseResultE} \overset{\text{def}}{=} \text{sum} (\text{Error} \rightarrow \langle \rangle, \text{SuccessP} \rightarrow \text{PrPat})
\]

seTLM Contexts

seTLM contexts, \( \Psi \), are of the form \( \langle A; \Psi \rangle \), where \( A \) is a TLM identifier expansion context and \( \Psi \) is a seTLM definition context.

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

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

**Definition B.3.8** (seTLM Definition Context Formation). \( \Delta \vdash \Psi \text{seTLMs} \iff \text{for each } x \rightsquigarrow \text{setlm}(\tau; e_{\text{parse}}) \in \Psi, \text{we have } \Delta \vdash \tau \text{ type and } \emptyset \emptyset \vdash e_{\text{parse}} : \text{parr}(\text{Body}; \text{ParseResultE}). \)

**Definition B.3.9** (seTLM Context Formation). \( \Delta \vdash \langle A; \Psi \rangle \text{seTLMctx} \iff \Delta \vdash \Psi \text{seTLMs} \text{ and for each } \hat{a} \rightsquigarrow x \in A \text{ we have } x \in \text{dom}(\Psi). \)

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

\[
\langle A \uplus \hat{a} \rightsquigarrow x; \Psi, x \rightsquigarrow \text{setlm}(\tau; e_{\text{parse}}) \rangle
\]
Typed Expression Expansion

spTLM Contexts

spTLM contexts, \( \Phi \), are of the form \( \langle A; \Phi \rangle \), where \( A \) is a TLM identifier expansion context, defined above, and \( \Phi \) is a spTLM definition context.

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

Definition B.3.10 (spTLM Definition Context Formation). \( \Delta \vdash \Phi \) spTLMs iff for each \( a \leftrightarrow \text{sptlm}(\tau; e_{\text{parse}}) \in \Phi \), we have \( \Delta \vdash \tau \) type and \( \emptyset \vdash e_{\text{parse}} : \text{parr}(\text{Body}; \text{ParseResultP}) \).

Definition B.3.11 (spTLM Context Formation). \( \Delta \vdash \langle A; \Phi \rangle \) spTLMctx iff \( \Delta \vdash \Phi \) spTLMs and for each \( \hat{a} \rightsquigarrow x \in A \) we have \( x \in \text{dom}(\Phi) \).

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

\[
\langle A \cup \hat{a} \rightsquigarrow x; \Phi, a \leftrightarrow \text{sptlm}(\tau; e_{\text{parse}}) \rangle
\]

Typed Expression Expansion

\[
\Delta \hat{\Gamma} \vdash_{\Phi} \hat{e} \rightsquigarrow e : \tau \quad \hat{e} \text{ has expansion } e \text{ of type } \tau
\]

\[
\Delta \hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau \vdash_{\Phi} \hat{x} \rightsquigarrow x : \tau
\]

\[
\Delta \vdash \hat{\tau} \rightsquigarrow \hat{\tau} \text{ type} \\
\Delta \hat{\Gamma} \vdash_{\Phi} \hat{e} \rightsquigarrow e : \tau
\]

\[
\Delta \hat{\Gamma} \vdash_{\Phi} \hat{e}_1 \rightsquigarrow e_1 : \tau_1 \\
\Delta \hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau_1 \vdash_{\Phi} \hat{e}_2 \rightsquigarrow e_2 : \tau_2
\]

\[
\Delta \hat{\Gamma} \vdash_{\Phi} \text{let val } \hat{x} = \hat{e}_1 \text{ in } \hat{e}_2 \rightsquigarrow \text{ap}(\lambda \{ \tau_1 \}(x.e_2); e_1) : \tau_2
\]

\[
\Delta \vdash \hat{\tau} \rightsquigarrow \hat{\tau} \text{ type} \\
\Delta \hat{\Gamma}, \hat{x} \rightsquigarrow x : \tau \vdash_{\Phi} \hat{e} \rightsquigarrow e : \tau' \]

\[
\Delta \hat{\Gamma} \vdash_{\Phi} \lambda \hat{x}. \hat{e} \rightsquigarrow \text{lam}(\tau)(x.e) : \text{parr}(\tau; \tau')
\]

\[
\Delta \hat{\Gamma} \vdash_{\Phi} \hat{e}_1 \rightsquigarrow e_1 : \text{parr}(\tau; \tau') \\
\Delta \hat{\Gamma} \vdash_{\Phi} \hat{e}_2 \rightsquigarrow e_2 : \tau
\]

\[
\Delta \hat{\Gamma} \vdash_{\Phi} \text{ap}(e_1; e_2) : \tau'
\]

\[
\hat{\Delta}, \hat{t} \rightsquigarrow t \text{ type } \hat{\Gamma} \vdash_{\Phi} \hat{e} \rightsquigarrow e : \tau
\]

\[
\Delta \hat{\Gamma} \vdash_{\Phi} \lambda \hat{t}. \hat{e} \rightsquigarrow \text{tlam}(t.e) : \text{all}(t.\tau)
\]
\[ \Delta \vdash \hat{\phi} \hat{\beta} \rightsquigarrow e : \text{all}(t.\tau) \quad \Delta \vdash \hat{\tau}' \rightsquigarrow \tau' \text{ type} \]  
(B.6g)

\[ \Delta \vdash \hat{\phi} \hat{\beta}[\hat{\tau}'] \rightsquigarrow \text{tap}(\hat{\tau}') (e) : [\tau'/t] \tau \]  
(B.6h)

\[ \Delta \vdash \hat{\phi} \hat{\beta} \rightsquigarrow e : \text{rec}(t.\tau)/t \]  
(B.6i)

\[ \Delta \vdash \hat{\phi} \hat{\beta} \rightsquigarrow \text{fold}(\hat{e}) \rightsquigarrow \text{fold}(e) : \text{rec}(t.\tau) \]  
(B.6j)

\[ \Delta \vdash \hat{\phi} \hat{\beta} \rightsquigarrow e : \text{rec}(t.\tau) \]  
(B.6k)

\[ \{ \Delta \vdash \hat{\phi} \hat{\beta}_i \rightsquigarrow e_i : \tau_i \}_{i \in L} \rightsquigarrow \text{prod}(\{ i \mapsto e_i \}_{i \in L}) \rightsquigarrow \text{prod}(\{ i \mapsto \tau_i \}_{i \in L}) \]  
(B.6l)

\[ \Delta \vdash \hat{\phi} \hat{\beta} \rightsquigarrow e : \text{sum}(\{ i \mapsto \tau_i \}_{i \in L}) \rightsquigarrow \text{sum}(\{ i \mapsto e_i \}_{i \in L}) : \tau \]  
(B.6m)

\[ \Delta \vdash \hat{\tau} \rightsquigarrow \tau \text{ type} \]

\[ \emptyset \emptyset \vdash e_{\text{parse}} : \text{parr}(\text{Body};\text{ParseResultE}) \quad \Delta \vdash \hat{\phi} \hat{\beta} \rightsquigarrow e_{\text{dep}} : \tau_{\text{dep}} \]  
(B.6n)

\[ \hat{\beta} = \langle G; \Gamma \rangle \quad \hat{\beta} = \langle G; \Gamma, x : \tau_{\text{dep}} \rangle \vdash \hat{\phi} \hat{\beta} \rightsquigarrow e : \tau' \]  
(B.6o)

\[ \hat{\beta} = \hat{\psi}', \hat{\alpha} \rightsquigarrow x \rightsquigarrow \text{setlm}(\tau; e_{\text{parse}}) \]  
(B.6p)
\[ \hat{\Delta} \vdash \hat{\tau} \rightsquigarrow \tau \text{ type} \quad \hat{\Delta} \vdash e_{\text{parse}} : \text{parr}(\text{Body}; \text{ParseResultP}) \]

\[ \frac{\hat{\Delta} \models \psi ; \Phi, \hat{x} \rightsquigarrow \text{sptlm}(\tau; e_{\text{parse}}) \quad \hat{e} \rightsquigarrow e : \tau'}{\hat{\Delta} \vdash \psi ; \Phi} \]

\[ \hat{\Delta} \vdash \psi ; \Phi \text{ notation } \hat{a} \text{ at } \tau \{ \text{pat parser } e_{\text{parse}} \} \text{ in } \hat{e} \rightsquigarrow e : \tau' \] (B.6q)

\[ \hat{\Delta} \vdash \psi ; \Phi \hat{p} \rightsquigarrow r : \tau \Rightarrow \tau' \quad \hat{p} \text{ has expansion } r \text{ taking values of type } \tau \text{ to values of type } \tau' \]

\[ \frac{\hat{\Delta} \vdash \psi ; \Phi \hat{p} \rightsquigarrow p : \tau \models \langle \hat{G}' ; \Gamma' \rangle \quad \hat{\Delta} \vdash \langle \hat{G} \uplus \hat{G}' ; \Gamma \uplus \Gamma' \rangle \vdash \psi ; \Phi \hat{e} \rightsquigarrow e : \tau'}{\hat{\Delta} \models \langle \hat{G} ; \Gamma \rangle \vdash \psi ; \Phi \text{ urule}(\hat{p} , \hat{e}) \rightsquigarrow \text{rule}(p , e) : \tau \Rightarrow \tau'} \] (B.7)

**Typed Pattern Expansion**

\[ \hat{\Delta} \models \psi ; \Phi \hat{p} \rightsquigarrow p : \tau \models \hat{\Gamma} \quad \hat{p} \text{ has expansion } p \text{ matching against } \tau \text{ generating hypotheses } \hat{\Gamma} \]

\[ \hat{\Delta} \vdash \psi ; \Phi \hat{x} \rightsquigarrow x : \tau \models \langle \hat{x} \rightsquigarrow x ; x : \tau \rangle \] (B.8a)

\[ \hat{\Delta} \vdash \psi ; \Phi \text{ wildp} : \tau \models \langle \emptyset ; \emptyset \rangle \] (B.8b)

\[ \hat{\Delta} \vdash \psi ; \Phi \hat{p} \rightsquigarrow p : [\text{rec}(t , \tau) / t] \tau \models \hat{\Gamma} \]

\[ \hat{\Delta} \vdash \psi ; \Phi \text{ fold}(\hat{p}) \rightsquigarrow \text{fold}(p) : \text{rec}(t , \tau) \models \hat{\Gamma} \] (B.8c)

\[ \tau = \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \quad \{\hat{\Delta} \models \psi ; \Phi \hat{p}_i \rightsquigarrow p_i : \tau_i \models \hat{\Gamma}_i\}_{i \in L} \]

\[ \hat{\Delta} \vdash \psi ; \Phi \langle\{i \mapsto \hat{p}_i\}_{i \in L}\rangle \rightsquigarrow \text{tplp}(\{i \mapsto p_i\}_{i \in L}) : \tau \models \uplus_{i \in L} \hat{\Gamma}_i \] (B.8d)

\[ \hat{\Delta} \vdash \psi ; \Phi \text{ inj}[\ell](\hat{p}) \rightsquigarrow \text{injp}[\ell](p) : \text{sum}(\{i \mapsto \tau_i\}_{i \in L} ; \ell \mapsto \tau) \models \hat{\Gamma} \] (B.8e)

\[ \Phi = \hat{\Phi}' , \hat{a} \rightsquigarrow _{\Phi} \quad \text{sptlm}(\tau ; e_{\text{parse}}) \quad \text{e}_{\text{parse}}(e_{\text{body}}) \downarrow \text{inj}[\text{SuccessP}](\text{e}_{\text{proto}}) \quad \text{e}_{\text{proto}} \uparrow_{\text{PrPat}} \hat{p} \]

\[ \text{seg}(\hat{p}) \text{ segments } b \quad \hat{\Delta} \vdash \psi ; \Phi \hat{a} \quad (b) \quad \rightsquigarrow p : \tau \models \hat{\Delta} ; \Phi ; b \hat{\Gamma} \] (B.8f)

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

\[ \langle \psi_{i \in L} G_i ; \cup_{i \in L} \Gamma_i \rangle \]
B.4 Proto-Expansion Validation

B.4.1 Syntax of Proto-Expansions

<table>
<thead>
<tr>
<th>Sort</th>
<th>Operational Form</th>
<th>Stylized Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrTyp</td>
<td>t ::= t</td>
<td>t</td>
<td>variable</td>
</tr>
<tr>
<td></td>
<td>prparr((\tilde{\tau}; \tilde{\tau}))</td>
<td>(\tilde{\tau} \rightarrow \tilde{\tau})</td>
<td>partial function</td>
</tr>
<tr>
<td></td>
<td>prall(t; (\tilde{\tau}))</td>
<td>(\forall t. \tilde{\tau})</td>
<td>polymorphic</td>
</tr>
<tr>
<td></td>
<td>prrec(t; (\tilde{\tau}))</td>
<td>(\mu t. \tilde{\tau})</td>
<td>recursive</td>
</tr>
<tr>
<td></td>
<td>prprod((i \mapsto \tilde{\tau}_i); (i \in L))</td>
<td>({{i \mapsto \tilde{\tau}_i}; i \in L})</td>
<td>labeled product</td>
</tr>
<tr>
<td></td>
<td>prsum((i \mapsto \tilde{\tau}_i); (i \in L))</td>
<td>([{i \mapsto \tilde{\tau}_i}; i \in L])</td>
<td>labeled sum</td>
</tr>
<tr>
<td></td>
<td>splicede[m; n]</td>
<td>splicede[m; n]</td>
<td>spliced type ref.</td>
</tr>
<tr>
<td>PrExp</td>
<td>(\tilde{e} ::= x)</td>
<td>x</td>
<td>variable</td>
</tr>
<tr>
<td></td>
<td>prasc{(\tilde{\tau})}((\tilde{e}))</td>
<td>(\tilde{e}: \tilde{\tau})</td>
<td>ascription</td>
</tr>
<tr>
<td></td>
<td>prletval((\tilde{e}); x; (\tilde{e}))</td>
<td>let val (x = \tilde{e}) in (\tilde{e})</td>
<td>value binding</td>
</tr>
<tr>
<td></td>
<td>prlam{x; (\tilde{e})}</td>
<td>(\lambda x. \tilde{e})</td>
<td>abstraction</td>
</tr>
<tr>
<td></td>
<td>prap((\tilde{e}); (\tilde{e}))</td>
<td>(\tilde{e}(\tilde{e}))</td>
<td>application</td>
</tr>
<tr>
<td></td>
<td>prtlam(t; (\tilde{e}))</td>
<td>(\Lambda t. \tilde{e})</td>
<td>type abstraction</td>
</tr>
<tr>
<td></td>
<td>prtap{(\tilde{\tau})}((\tilde{e}))</td>
<td>(\tilde{e}[\tilde{\tau}])</td>
<td>type application</td>
</tr>
<tr>
<td></td>
<td>prfold((\tilde{e}))</td>
<td>fold((\tilde{e}))</td>
<td>fold</td>
</tr>
<tr>
<td></td>
<td>prunfold((\tilde{e}))</td>
<td>unfold((\tilde{e}))</td>
<td>unfold</td>
</tr>
<tr>
<td></td>
<td>prtpl((i \mapsto \tilde{e}_i); (i \in L))</td>
<td>({{i \mapsto \tilde{e}_i}; i \in L})</td>
<td>labeled tuple</td>
</tr>
<tr>
<td></td>
<td>prprj{(\ell)}((\tilde{e}))</td>
<td>(\tilde{e} \cdot \ell)</td>
<td>projection</td>
</tr>
<tr>
<td></td>
<td>prinj{(\ell)}((\tilde{e}))</td>
<td>inj{(\ell)}((\tilde{e}))</td>
<td>injection</td>
</tr>
<tr>
<td></td>
<td>prcase((\tilde{e}); ({i \mapsto \tilde{x}_i, \tilde{e}_i}; i \in L))</td>
<td>case (\tilde{e}) ({i \mapsto \tilde{x}_i, \tilde{e}_i}; i \in L)</td>
<td>case analysis</td>
</tr>
<tr>
<td></td>
<td>splicede[m; n; (\tilde{\tau})]</td>
<td>splicede[m; n; (\tilde{\tau})]</td>
<td>spliced expr. ref.</td>
</tr>
<tr>
<td>PrRule</td>
<td>(\tilde{r} ::= \text{prrule}(\text{p}; (\tilde{e})))</td>
<td>(\text{p} \Rightarrow \tilde{e})</td>
<td>rule</td>
</tr>
<tr>
<td>PrPat</td>
<td>(\tilde{p} ::= \text{prwild})</td>
<td>_</td>
<td>wildcard pattern</td>
</tr>
<tr>
<td></td>
<td>prfoldp((\tilde{p}))</td>
<td>fold((\tilde{p}))</td>
<td>fold pattern</td>
</tr>
<tr>
<td></td>
<td>prtplp{(L)}(({i \mapsto \tilde{p}_i}; i \in L))</td>
<td>({{i \mapsto \tilde{p}_i}; i \in L})</td>
<td>labeled tuple pattern</td>
</tr>
<tr>
<td></td>
<td>prinjp{(\ell)}((\tilde{p}))</td>
<td>inj{(\ell)}((\tilde{p}))</td>
<td>injection pattern</td>
</tr>
<tr>
<td></td>
<td>splicepred[m; n; (\tilde{\tau})]</td>
<td>splicepred[m; n; (\tilde{\tau})]</td>
<td>spliced pattern ref.</td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
\mathcal{P}(t) &= t \\
\mathcal{P}(\text{parr}(\tau_1; \tau_2)) &= \text{prparr}(\mathcal{P}(\tau_1); \mathcal{P}(\tau_2)) \\
\mathcal{P}(\text{all}(t; \tau)) &= \text{prall}(t; \mathcal{P}(\tau)) \\
\mathcal{P}(\text{rec}(t; \tau)) &= \text{prrec}(t; \mathcal{P}(\tau)) \\
\mathcal{P}(\text{prod}(\{i \mapsto \tau_i\}_{i \in L})) &= \text{prprod}(\{i \mapsto \mathcal{P}(\tau_i)\}_{i \in L}) \\
\mathcal{P}(\text{sum}(\{i \mapsto \tau_i\}_{i \in L})) &= \text{prsum}(\{i \mapsto \mathcal{P}(\tau_i)\}_{i \in L})
\end{align*}
\]

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

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

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

\[
\mathcal{P}(\text{rule}(p.e)) = \text{prrule}(p; \mathcal{P}(e))
\]

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:

\[
\begin{align*}
\mathcal{P}(\text{wildp}) &= \text{prwildp} \\
\mathcal{P}(\text{foldp}(p)) &= \text{prfoldp}(\mathcal{P}(p)) \\
\mathcal{P}(\text{tplp}(\{i \mapsto p_i\}_{i \in L})) &= \text{prtplp}[L](\{i \mapsto \mathcal{P}(p_i)\}_{i \in L}) \\
\mathcal{P}(\text{injp}[\ell](p)) &= \text{prinjp}[\ell](\mathcal{P}(p))
\end{align*}
\]

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, $\downarrow_{\text{PrExpr}} e$. An inverse mapping is defined by the proto-expression decoding judgement, $e \uparrow_{\text{PrExpr}} \hat{e}$.
Judgement Form | Description
--- | ---
\( \hat{e} \downarrow_{\text{PrExpr}} e \) | \( \hat{e} \) has encoding \( e \)
\( e \uparrow_{\text{PrExpr}} \hat{e} \) | \( e \) has decoding \( \hat{e} \)

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

**Condition B.4.1 (Proto-Expression Isomorphism).**

1. For every \( \hat{e} \), we have \( \hat{e} \downarrow_{\text{PrExpr}} e_{\text{proto}} \), for some \( e_{\text{proto}} \) such that \( \vdash e_{\text{proto}} : \text{PrExpr} \) and \( e_{\text{proto}} \) val.

2. If \( \vdash e_{\text{proto}} : \text{PrExpr} \) and \( e_{\text{proto}} \) val then \( e_{\text{proto}} \uparrow_{\text{PrExpr}} \hat{e} \) for some \( \hat{e} \).

3. If \( \hat{e} \downarrow_{\text{PrExpr}} e_{\text{proto}} \) then \( e_{\text{proto}} \uparrow_{\text{PrExpr}} \hat{e} \).

4. If \( \vdash e_{\text{proto}} : \text{PrExpr} \) and \( e_{\text{proto}} \) val and \( e_{\text{proto}} \uparrow_{\text{PrExpr}} \hat{e} \) then \( \hat{e} \downarrow_{\text{PrExpr}} e_{\text{proto}} \).

5. If \( \hat{e} \downarrow_{\text{PrExpr}} e_{\text{proto}} \) and \( \hat{e} \downarrow_{\text{PrExpr}} e'_{\text{proto}} \) then \( e_{\text{proto}} = e'_{\text{proto}} \).

6. If \( \vdash e_{\text{proto}} : \text{PrExpr} \) and \( e_{\text{proto}} \) val and \( e_{\text{proto}} \uparrow_{\text{PrExpr}} \hat{e} \) and \( e_{\text{proto}} \uparrow_{\text{PrExpr}} \hat{e}' \) then \( \hat{e} = \hat{e}' \).

Proto-Pattern Encoding and Decoding

The type abbreviated \( \text{PrPat} \) classifies encodings of proto-patterns. The mapping from proto-patterns to values of type \( \text{PrPat} \) is defined by the proto-pattern encoding judgement, \( \hat{p} \downarrow_{\text{PrPat}} p \). An inverse mapping is defined by the proto-expression decoding judgement, \( p \uparrow_{\text{PrPat}} \hat{p} \).

<table>
<thead>
<tr>
<th>Judgement Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{p} \downarrow_{\text{PrPat}} p )</td>
<td>( \hat{p} ) has encoding ( p )</td>
</tr>
<tr>
<td>( p \uparrow_{\text{PrPat}} \hat{p} )</td>
<td>( p ) has decoding ( \hat{p} )</td>
</tr>
</tbody>
</table>

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

**Condition B.4.2 (Proto-Pattern Isomorphism).**

1. For every \( \hat{p} \), we have \( \hat{p} \downarrow_{\text{PrPat}} e_{\text{proto}} \), for some \( e_{\text{proto}} \) such that \( \vdash e_{\text{proto}} : \text{PrPat} \) and \( e_{\text{proto}} \) val.

2. If \( \vdash e_{\text{proto}} : \text{PrPat} \) and \( e_{\text{proto}} \) val then \( e_{\text{proto}} \uparrow_{\text{PrPat}} \hat{p} \) for some \( \hat{p} \).

3. If \( \hat{p} \downarrow_{\text{PrPat}} e_{\text{proto}} \) then \( e_{\text{proto}} \uparrow_{\text{PrPat}} \hat{p} \).

4. If \( \vdash e_{\text{proto}} : \text{PrPat} \) and \( e_{\text{proto}} \) val and \( e_{\text{proto}} \uparrow_{\text{PrPat}} \hat{p} \) then \( \hat{p} \downarrow_{\text{PrPat}} e_{\text{proto}} \).

5. If \( \hat{p} \downarrow_{\text{PrPat}} e_{\text{proto}} \) and \( \hat{p} \downarrow_{\text{PrPat}} e'_{\text{proto}} \) then \( e_{\text{proto}} = e'_{\text{proto}} \).
6. If $\vdash e_{\text{proto}} : \text{PrPat}$ and $e_{\text{proto}} \downarrow_{\text{PrPat}} \hat{p}$ and $e_{\text{proto}} \downarrow_{\text{PrPat}} \hat{p}'$ then $\hat{p} = \hat{p}'$.

## Segmentations

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Segment Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{seg}(t)$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prparr}(\hat{\tau}_1; \hat{\tau}_2))$</td>
<td>$\text{seg}(\hat{\tau}_1) \cup \text{seg}(\hat{\tau}_2)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prall}(t; \hat{\tau}))$</td>
<td>$\text{seg}(\hat{\tau})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prrec}(t; \hat{\tau}))$</td>
<td>$\text{seg}(\hat{\tau})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prprod}([i \mapsto \hat{\tau}<em>i]</em>{i \in L}))$</td>
<td>$\bigcup_{i \in L} \text{seg}(\hat{\tau}_i)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prsum}([i \mapsto \hat{\tau}<em>i]</em>{i \in L}))$</td>
<td>$\bigcup_{i \in L} \text{seg}(\hat{\tau}_i)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{splicedt}[m; n])$</td>
<td>${\text{splicedt}[m; n]}$</td>
</tr>
<tr>
<td>$\text{seg}(x)$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prasc}{\hat{\tau}}([\hat{e}])$</td>
<td>$\text{seg}(\hat{\tau}) \cup \text{seg}(\hat{e})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prletval}(\hat{e}_1; x. \hat{e}_2))$</td>
<td>$\text{seg}(\hat{e}_1) \cup \text{seg}(\hat{e}_2)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prlam}{\hat{\tau}}(x. \hat{e}))$</td>
<td>$\text{seg}(\hat{\tau}) \cup \text{seg}(\hat{e})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prap}(\hat{\tau}_1; \hat{\tau}_2))$</td>
<td>$\text{seg}(\hat{\tau}_1) \cup \text{seg}(\hat{\tau}_2)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prtlam}(\hat{\tau}_1; \hat{\tau}_2))$</td>
<td>$\text{seg}(\hat{\tau}_1) \cup \text{seg}(\hat{\tau}_2)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prtap}{\hat{\tau}}([\hat{e}])$</td>
<td>$\text{seg}(\hat{e}) \cup \text{seg}(\hat{\tau})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prfold}(\hat{\tau}))$</td>
<td>$\text{seg}(\hat{\tau})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prunfold}(\hat{\tau}))$</td>
<td>$\text{seg}(\hat{\tau})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prpll}([i \mapsto x_i. \hat{e}<em>i]</em>{i \in L}))$</td>
<td>$\bigcup_{i \in L} \text{seg}(\hat{e}_i)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prprj}<a href="%5Chat%7Be%7D">\hat{\tau}</a>)$</td>
<td>$\text{seg}(\hat{e})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prinj}<a href="%5Chat%7Be%7D">\hat{\tau}</a>)$</td>
<td>$\text{seg}(\hat{e})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prcase}(\hat{e}; [i \mapsto x_i. \hat{e}<em>i]</em>{i \in L}))$</td>
<td>$\text{seg}(\hat{e}) \cup \bigcup_{i \in L} \text{seg}(\hat{e}_i)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{splicede}[m; n; \hat{\tau}])$</td>
<td>${\text{splicede}[m; n; \hat{\tau}]} \cup \text{seg}(\hat{\tau})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prmatch}(\hat{e}; [\hat{\tau}<em>i]</em>{1 \leq i \leq n}))$</td>
<td>$\text{seg}(\hat{e}) \cup \bigcup_{1 \leq i \leq n} \text{seg}(\hat{\tau}_i)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prrule}(p; \hat{e}))$</td>
<td>$\text{seg}(\hat{e})$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Segment Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{seg}(\text{prwildp})$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prfoldp}(\hat{p}))$</td>
<td>$\text{seg}(\hat{p})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prplp}[L]([i \mapsto \hat{p}<em>i]</em>{i \in L}))$</td>
<td>$\bigcup_{i \in L} \text{seg}(\hat{p}_i)$</td>
</tr>
<tr>
<td>$\text{seg}(\text{prinjp}<a href="%5Chat%7Bp%7D">\hat{\ell}</a>)$</td>
<td>$\text{seg}(\hat{p})$</td>
</tr>
<tr>
<td>$\text{seg}(\text{splicep}[m; n; \hat{\tau}])$</td>
<td>${\text{splicep}[m; n; \hat{\tau}]} \cup \text{seg}(\hat{\tau})$</td>
</tr>
</tbody>
</table>

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

**Definition B.4.3 (Segmentation Validity).** $\psi$ segments $b$ iff

1. For each $spliced[t][m; n] \in \psi$, all of the following hold:
   
   (a) $0 \leq m \leq n < \|b\|
   
   (b) For each $spliced[t][m'; n'] \in \psi$, either
      
      i. $m = m'$ and $n = n'$; or
      
      ii. $n' < m - 1$; or
      
      iii. $m' > n + 1$
   
   (c) For each $spliced[e][m'; n'; \tau] \in \psi$, either
      
      i. $n' < m - 1$; or
      
      ii. $m' > n + 1$

2. For each $spliced[e][m; n; \tau] \in \psi$, all of the following hold:
   
   (a) $0 \leq m \leq n < \|b\|
   
   (b) For each $spliced[t][m'; n'] \in \psi$, either
      
      i. $n' < m - 1$; or
      
      ii. $m' > n + 1$
   
   (c) For each $spliced[e][m'; n'; \tau'] \in \psi$, either
      
      i. $m = m'$ and $n = n'$ and $\tau = \tau'$; or
      
      ii. $n' < m - 1$; or
      
      iii. $m' > n + 1$

3. For each $spliced[p][m; n; \tau] \in \psi$, all of the following hold:
   
   (a) $0 \leq m \leq n < \|b\|
   
   (b) For each $spliced[t][m'; n'] \in \psi$, either
      
      i. $n' < m - 1$; or
      
      ii. $m' > n + 1$
   
   (c) For each $spliced[e][m'; n'; \tau'] \in \psi$, either
      
      i. $n' < m - 1$; or
      
      ii. $m' > n + 1$

(d) For each $spliced[p][m'; n'; \tau'] \in \psi$, either
i. \( m = m' \) and \( n = n' \) and \( \tau = \hat{\tau}' \); or

ii. \( n' < m - 1 \); or

iii. \( m' > n + 1 \)

### B.4.2 Proto-Type Validation

**Type splicing scenes**, \( \mathcal{T} \), are of the form \( \hat{\Delta}; \; b \).

\[
\Delta \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type} \quad \hat{\tau} \text{ has well-formed expansion } \tau
\]

\[
\frac{\Delta, t \text{ type } \vdash_\mathcal{T} t \rightsquigarrow t \text{ type}}{\Delta, t \text{ type } \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type}} \quad \text{(B.9a)}
\]

\[
\begin{align*}
\Delta & \vdash_\mathcal{T} \hat{\tau}_1 \rightsquigarrow \tau_1 \text{ type} & \Delta & \vdash_\mathcal{T} \hat{\tau}_2 \rightsquigarrow \tau_2 \text{ type} \\
\Delta & \vdash_\mathcal{T} \text{prparr}(\hat{\tau}_1; \hat{\tau}_2) \rightsquigarrow \text{parr}(\tau_1; \tau_2) \text{ type} \\
\Delta & \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type} & \Delta & \vdash_\mathcal{T} \text{prall}(t. \hat{\tau}) \rightsquigarrow \text{all}(t. \tau) \text{ type} \\
\Delta & \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type} & \Delta & \vdash_\mathcal{T} \text{prrec}(t. \hat{\tau}) \rightsquigarrow \text{rec}(t. \tau) \text{ type}
\end{align*}
\]

\[
\frac{\{ \Delta \vdash_\mathcal{T} \hat{\tau}_i \rightsquigarrow \tau_i \text{ type} \}_{i \in L}}{\Delta \vdash_\mathcal{T} \text{prprod}(\{i \mapsto \hat{\tau}_i\}_{i \in L}) \rightsquigarrow \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \text{ type}} \quad \text{(B.9e)}
\]

\[
\frac{\{ \Delta \vdash_\mathcal{T} \hat{\tau}_i \rightsquigarrow \tau_i \text{ type} \}_{i \in L}}{\Delta \vdash_\mathcal{T} \text{prsum}(\{i \mapsto \hat{\tau}_i\}_{i \in L}) \rightsquigarrow \text{sum}(\{i \mapsto \tau_i\}_{i \in L}) \text{ type}} \quad \text{(B.9f)}
\]

\[
\text{parseUTyp(subseq}(b; m; n)) = \hat{\tau} \quad \langle D; \Delta_{\text{app}} \rangle \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type} \quad \Delta \cap \Delta_{\text{app}} = \emptyset
\]

\[
\frac{\langle D; \Delta_{\text{app}} \rangle \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type}}{\Delta \vdash_{\langle D; \Delta_{\text{app}} \rangle ; b} \text{splicedt}[m; n] \rightsquigarrow \tau \text{ type}} \quad \text{(B.9g)}
\]

### B.4.3 Proto-Expression Validation

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

\[
\text{ts}(\hat{\Delta}; \; \hat{\Gamma}; \; \hat{\Psi}; \; \hat{\Phi}; \; b) = \hat{\Delta}; \; b
\]

\[
\Delta \Gamma \vdash_\mathcal{E} \hat{e} \rightsquigarrow e : \tau \quad \hat{e} \text{ has expansion } e \text{ of type } \tau
\]

\[
\frac{\Delta \Gamma \vdash_\mathcal{E} x : \tau \vdash_\mathcal{E} x \rightsquigarrow x : \tau}{\Delta \Gamma \vdash_\mathcal{E} \hat{\tau} \rightsquigarrow \tau \text{ type} \quad \Delta \Gamma \vdash_\mathcal{E} \hat{e} \rightsquigarrow e : \tau}
\]

\[
\frac{\Delta \Gamma \vdash_\mathcal{E} \text{prasc}(\hat{\tau})(\hat{e}) \rightsquigarrow e : \tau}{\Delta \Gamma \vdash_\mathcal{E} \hat{\tau} \rightsquigarrow \tau \text{ type} \quad \Delta \Gamma \vdash_\mathcal{E} \hat{e} \rightsquigarrow e : \tau}
\]

\[
\text{parseUTyp(subseq}(b; m; n)) = \hat{\tau} \quad \langle D; \Delta_{\text{app}} \rangle \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type} \quad \Delta \cap \Delta_{\text{app}} = \emptyset
\]

\[
\frac{\langle D; \Delta_{\text{app}} \rangle \vdash_\mathcal{T} \hat{\tau} \rightsquigarrow \tau \text{ type}}{\Delta \vdash_{\langle D; \Delta_{\text{app}} \rangle ; b} \text{splicedt}[m; n] \rightsquigarrow \tau \text{ type}} \quad \text{(B.9g)}
\]
\[ \Delta \Gamma \vdash^{E} \hat{e} \rightarrow e : \tau \quad \Delta \Gamma, x : \tau \vdash \hat{e} \rightarrow e_2 : \tau_2 : \]  

(B.10c)

\[ \Delta \Gamma \vdash^{E} \text{prletval}(\hat{e}; x.\hat{e}_2) \rightarrow \text{ap}(\lambda \{\tau_1\}(x.\hat{e}_2); e_1) : \tau_2 \]  

(B.10d)

\[ \Delta \Gamma, x : \tau \vdash^{E} \hat{e} \rightarrow e : \tau' \]  

(B.10e)

\[ \Delta \Gamma \vdash^{E} \text{prlam}(\hat{e})(x.\hat{e}) \rightarrow \text{lam}(\hat{e})(x.e) : \text{parr}(\tau; \tau') \]  

(B.10f)

\[ \Delta \Gamma \vdash^{E} \hat{e}_1 \rightarrow e_1 : \text{parr}(\tau; \tau') \quad \Delta \Gamma \vdash^{E} \hat{e}_2 \rightarrow e_2 : \tau \]  

(B.10g)

\[ \Delta, t \text{ type } \Delta \vdash^{E} \hat{e} \rightarrow e : \tau \]  

(B.10h)

\[ \Delta \Gamma \vdash^{E} \text{prtlam}(t.\hat{e}) \rightarrow \text{tlam}(t.e) : \text{all}(t.\tau) \]  

(B.10i)

\[ \Delta \Gamma \vdash^{E} \hat{e} \rightarrow e : \text{all}(t.\tau) \quad \Delta \vdash^{E} \tau' \rightarrow \tau' \text{ type} \]  

(B.10j)

\[ \Delta \Gamma \vdash^{E} \text{prtap}(\hat{e}') \rightarrow \text{tap}(\hat{e}')(e) : [\tau'/t] \tau \]  

(B.10k)

\[ \Delta \Gamma \vdash^{E} \hat{e} \rightarrow e : [\text{rec}(t.\tau)/t] \tau \]  

(B.10l)

\[ \Delta \Gamma \vdash^{E} \text{prj}[\ell](\hat{e}) \rightarrow \text{prj}[\ell](e) : \tau \]  

(B.10m)

\[ \Delta \Gamma \vdash^{E} \text{prcase}(\hat{e}; \{i \mapsto \hat{e}_i\}_{i \in \mathcal{L}}) \rightarrow \text{case}(e; \{i \mapsto x_i\}_{i \in \mathcal{L}}) : \tau \]  

(B.10n)

\[ \emptyset \vdash^{E} \hat{r} \rightarrow \tau \text{ type} \quad \mathcal{E} = \{D; \Delta_{\text{app}}; \langle G; \Gamma_{\text{app}} \rangle; \Psi; \Phi; b \} \]  

parseUEExp(subseq(b; m; n)) = \hat{e} \rightarrow \{D; \Delta_{\text{app}}; \langle G; \Gamma_{\text{app}} \rangle \vdash \Psi; \Phi \hat{e} \rightarrow e : \tau \}

\[ \emptyset \cap \Delta_{\text{app}} = \emptyset \quad \text{dom}(\Gamma) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset \]  

(B.10o)

\[ \Delta \Gamma \vdash^{E} \text{splicede}[m; n; \hat{r}] \rightarrow e : \tau \]  

\[ \Delta \Gamma \vdash^{E} \hat{r}_i \rightarrow e : \tau_i \quad \{\Delta \Gamma \vdash^{E} \hat{r}_i \rightarrow r_i : \tau \Rightarrow \tau'_i\}_{1 \leq i \leq n} \]  

\[ \Delta \Gamma \vdash^{E} \text{prmatch}(\hat{r}_i; \{r_i\}_{1 \leq i \leq n}) \rightarrow \text{match}(e; \{r_i\}_{1 \leq i \leq n}) : \tau' \]  

(B.10p)
\[ \Delta \Gamma \vdash^E \tau \leadsto r : \tau \Rightarrow \tau' \]
\[ \Delta \cup \Delta_{app} \vdash p : \tau \dashv \Gamma' \quad \Delta \Gamma \cup \Gamma' \vdash^E e \leadsto e : \tau' \]
\[ \Delta \Gamma \vdash^E \text{prrule}(p, e) \leadsto \text{rule}(p, e) : \tau \Rightarrow \tau' \]
(B.11)

### B.4.4 Proto-Pattern Validation

*Pattern splicing scenes, \( \mathcal{P} \), are of the form \( \hat{\Delta}; \hat{\Phi}; b \).*

\[ \hat{p} \leadsto p : \tau \vdash^P \hat{\Gamma} \]
\[ \hat{p} \leadsto p : \tau \vdash^P \langle \emptyset; \emptyset \rangle \]
(B.12a)

\[ \hat{p} \leadsto p : [\text{rec}(t, \tau) / t] \tau \vdash^P \hat{\Gamma} \]
(B.12b)

\[ \hat{p} \leadsto p : \tau \vdash^P \langle \emptyset \rangle \hat{\Gamma} \]
(B.12c)

\[ \hat{p} \leadsto p : \tau \vdash^P \langle \emptyset \rangle \hat{\Gamma} \]
(B.12d)

\[ \emptyset \vdash \hat{\Delta}; b \hat{\tau} \leadsto \tau \text{ type} \quad \text{parseUPat}(\text{subseq}(b; m; n)) = \hat{p} \quad \hat{\Delta} \vdash \hat{\Phi} \hat{p} \leadsto p : \tau \vdash^P \hat{\Gamma} \]
(B.12e)

### B.5 Metatheory

#### B.5.1 Type Expansion

**Lemma B.5.1 (Type Expansion).** If \( \langle D; \Delta \rangle \vdash \hat{\tau} \leadsto \tau \text{ type} \) then \( \Delta \vdash \tau \text{ type} \).

*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). \( \Box \)

**Lemma B.5.2 (Proto-Type Validation).** If \( \Delta \vdash \langle D; \Delta_{app} \rangle; b \hat{\tau} \leadsto \tau \text{ type} \) and \( \Delta \cap \Delta_{app} = \emptyset \) then \( \Delta \cup \Delta_{app} \vdash \tau \text{ type} \).

*Proof.* By rule induction over Rules (B.9).
Case (B.9a).

(1) $\Delta = \Delta', t \text{ type}$ by assumption
(2) $\tilde{\tau} = t$ by assumption
(3) $\tau = t$ by assumption
(4) $\Delta', t \text{ type} \vdash t \text{ type}$ by Rule (B.1a)
(5) $\Delta', t \text{ type} \cup \Delta_{\text{app}} \vdash t \text{ type}$ by Lemma B.2.2 over $\Delta_{\text{app}}$ to (4)

Case (B.9b).

(1) $\tilde{\tau} = \text{prparr}(\tilde{\tau}_1; \tilde{\tau}_2)$ by assumption
(2) $\tau = \text{parr}(\tau_1; \tau_2)$ by assumption
(3) $\Delta \vdash (D; \Delta_{\text{app}}); b \tilde{\tau}_1 \leadsto \tau_1 \text{ type}$ by assumption
(4) $\Delta \vdash (D; \Delta_{\text{app}}); b \tilde{\tau}_2 \leadsto \tau_2 \text{ type}$ by assumption
(5) $\Delta \cup \Delta_{\text{app}} \vdash \tau_1 \text{ type}$ by IH on (3)
(6) $\Delta \cup \Delta_{\text{app}} \vdash \tau_2 \text{ type}$ by IH on (4)
(7) $\Delta \cup \Delta_{\text{app}} \vdash \text{parr}(\tau_1; \tau_2) \text{ type}$ by Rule (B.1b) on (5) and (6)

Case (B.9c).

(1) $\tilde{\tau} = \text{prall}(t, \tilde{\tau}')$ by assumption
(2) $\tau = \text{all}(t, \tau')$ by assumption
(3) $\Delta, t \text{ type} \vdash (D; \Delta_{\text{app}}); b \tilde{\tau}' \leadsto \tau' \text{ type}$ 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 $\Delta_{\text{app}}$ on (4)
(6) $\Delta \cup \Delta_{\text{app}} \vdash \text{all}(t, \tau') \text{ type}$ by Rule (B.1c) on (5)

Case (B.9d).

(1) $\tilde{\tau} = \text{prrec}(t, \tilde{\tau}')$ by assumption
(2) $\tau = \text{rec}(t, \tau')$ by assumption
(3) $\Delta, t \text{ type} \vdash \text{app}, b \tilde{\tau}' \leadsto \tau' \text{ type}$ 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 \( \Delta_{\text{app}} \) on (4)

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

Case (B.9e).

1. \( \hat{\tau} = \text{prprod}(\{i \mapsto \hat{\tau}_i\}_{i \in L}) \) by assumption
2. \( \tau = \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \) by assumption
3. \( \{\Delta \vdash \Delta_{\text{app}}; b \hat{\tau}_i \rightsquigarrow \tau_i \text{ type}\}_{i \in L} \) by assumption
4. \( \{\Delta \cup \Delta_{\text{app}} \vdash \tau_i \text{ type}\}_{i \in L} \) by IH over (3)
5. \( \Delta \cup \Delta_{\text{app}} \vdash \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \text{ type} \) by Rule (B.1e) on (4)

Case (B.9f).

1. \( \hat{\tau} = \text{prsum}(\{i \mapsto \hat{\tau}_i\}_{i \in L}) \) by assumption
2. \( \tau = \text{sum}(\{i \mapsto \tau_i\}_{i \in L}) \) by assumption
3. \( \{\Delta \vdash \Delta_{\text{app}}; b \hat{\tau}_i \rightsquigarrow \tau_i \text{ type}\}_{i \in L} \) by assumption
4. \( \{\Delta \cup \Delta_{\text{app}} \vdash \tau_i \text{ type}\}_{i \in L} \) by IH over (3)
5. \( \Delta \cup \Delta_{\text{app}} \vdash \text{sum}(\{i \mapsto \tau_i\}_{i \in L}) \text{ type} \) by Rule (B.1f) on (4)

Case (B.9g).

1. \( \hat{\tau} = \text{splicedt}[m; n] \) by assumption
2. \( \text{parseUTyp}(\text{subseq}(b; m; n)) = \hat{\tau} \) by assumption
3. \( \langle D; \Delta_{\text{app}} \rangle \vdash \hat{\tau} \rightsquigarrow \tau \text{ type} \) by assumption
4. \( \Delta \cap \Delta_{\text{app}} = \emptyset \) by assumption
5. \( \Delta_{\text{app}} \vdash \tau \text{ type} \) by Lemma B.5.1 on (3)
6. \( \Delta \cup \Delta_{\text{app}} \vdash \tau \text{ type} \) by Lemma B.2.2 over \( \Delta \) on (5) and exchange over \( \Delta \)

\( \square \)
B.5.2 Typed Pattern Expansion

**Theorem B.5.3 (Typed Pattern Expansion).**

1. If \( \langle D; \Delta \rangle \vdash \langle A; \Phi \rangle \hat{p} \rightsquigarrow p : \tau \mid\mid \langle G; \Gamma \rangle \) then \( p : \tau \mid\mid \Gamma \).

2. If \( \hat{p} \rightsquigarrow p : \tau \mid\mid \langle D; \Delta \rangle; \langle A; \Phi \rangle; b \langle G; \Gamma \rangle \) then \( p : \tau \mid\mid \Gamma \).

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

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

**Case (B.8a).**

(1) \( \hat{p} = \hat{x} \) by assumption

(2) \( p = x \) by assumption

(3) \( \Gamma = x : \tau \) by assumption

(4) \( x : \tau \mid\mid x : \tau \) by Rule (B.4a)

**Case (B.8b).**

(1) \( p = \text{wildp} \) by assumption

(2) \( \Gamma = \emptyset \) by assumption

(3) \( \text{wildp} : \tau \mid\mid \emptyset \) by Rule (B.4b)

**Case (B.8c).**

(1) \( \hat{p} = \text{fold}(\hat{p}') \) by assumption

(2) \( p = \text{foldp}(p') \) by assumption

(3) \( \tau = \text{rec}(t.\tau') \) by assumption

(4) \( \hat{\Delta} \vdash \hat{\Phi} \hat{p}' \rightsquigarrow p' : [\text{rec}(t.\tau')/t]\tau' \mid\mid \hat{\Gamma} \) by assumption

(5) \( p' : [\text{rec}(t.\tau')/t]\tau' \mid\mid \Gamma \) by IH, part 1 on (4)

(6) \( \text{foldp}(p') : \text{rec}(t.\tau') \mid\mid \Gamma \) by Rule (B.4c) on (5)

**Case (B.8d).**

(1) \( \hat{p} = \{i \mapsto \hat{p}_i\}_{i \in L} \) by assumption

(2) \( p = \text{tplp}(\{i \mapsto p_i\}_{i \in L}) \) by assumption

(3) \( \tau = \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \) by assumption

(4) \( \{\hat{\Delta} \vdash \hat{\Phi} \hat{p}_i \rightsquigarrow p_i : \tau_i \mid\mid \langle G_i; \Gamma_i \rangle\}_{i \in L} \) by assumption

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

(6) \( \{p_i : \tau_i \mid\mid \Gamma_i\}_{i \in L} \) by IH, part 1 over (4)

(7) \( \text{tplp}(\{i \mapsto p_i\}_{i \in L}) : \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \mid\mid \cup_{i \in L} \Gamma_i \) by Rule (B.4d) on (6)
Case (B.8e).

(1) \( \hat{p} = \text{inj}[\ell](\hat{p}') \) by assumption
(2) \( p = \text{inj}[\ell](p') \) by assumption
(3) \( \tau = \text{sum}(\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau') \) by assumption
(4) \( \hat{\Delta} \vdash \hat{\Phi} \hat{p}' \Rightarrow p' : \tau' \Downarrow \hat{\Gamma} \) by assumption
(5) \( p' : \tau' \Downarrow \Gamma \) by IH, part 1 on (4)
(6) \( \text{injp}[\ell](p') : \text{sum}(\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau') \Downarrow \Gamma \) by Rule (B.4e) on (5)

Case (B.8f).

(1) \( \hat{p} = \hat{a}'(b)' \) by assumption
(2) \( \mathcal{A} = \mathcal{A}', \hat{a} \leadsto x \) by assumption
(3) \( \Phi = \Phi', a \mapsto \text{sptlm}(\tau; e_{\text{parse}}) \) by assumption
(4) \( b \Downarrow \text{Body} e_{\text{body}} \) by assumption
(5) \( e_{\text{parse}}(e_{\text{body}}) \Downarrow \text{inj}[\text{SuccessP}](e_{\text{proto}}) \) by assumption
(6) \( e_{\text{proto}} \uparrow \text{PrPat} \hat{p} \) by assumption
(7) \( \hat{p} \Rightarrow p : \tau \Downarrow [\hat{\Delta}; (\mathcal{A}; \Phi); b] (\mathcal{G}; \Gamma) \) by assumption
(8) \( p : \tau \Downarrow \Gamma \) 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 \( \Phi = \langle \mathcal{A}; \Phi \rangle \).

Case (B.12a).

(1) \( p = \text{wildp} \) by assumption
(2) \( \Gamma = \emptyset \) by assumption
(3) \( \text{wildp} : \tau \Downarrow \emptyset \) by Rule (B.4b)

Case (B.12b).

(1) \( \hat{p} = \text{prfoldp}(\hat{p}') \) by assumption
(2) \( p = \text{foldp}(p') \) by assumption
(3) \( \tau = \text{rec}(t.\tau') \) by assumption
(4) \( \hat{p}' \Rightarrow p' : [\text{rec}(t.\tau') / t] \tau' \Downarrow [\hat{\Delta}; \hat{\Phi}; b] \hat{\Gamma} \) by assumption
(5) \( p' : [\text{rec}(t.\tau') / t] \tau' \Downarrow \Gamma \) by IH, part 2 on (4)
(6) \( \text{foldp}(p') : \text{rec}(t.\tau') \Downarrow \Gamma \) by Rule (B.4c) on (5)

Case (B.12c).

(1) \( \hat{p} = \text{prtplp}[L](\{i \mapsto \hat{p}_i\}_{i \in L}) \) by assumption
(2) \( p = \text{tplp}(\{i \mapsto p_i\}_{i \in L}) \) by assumption
(3) \( \tau = \text{prod}(\{i \mapsto \tau_i\}_{i \in L}) \) by assumption
\begin{align*}
(4) \quad \{ \hat{p}_i \leadsto p_i : \tau_i \parallel \hat{\Delta}; \hat{\Phi}; b \langle G_i; \Gamma_i \rangle \}_{i \in L} & \quad \text{by assumption} \\
(5) \quad \Gamma = \bigcup_{i \in L} \Gamma_i & \quad \text{by assumption} \\
(6) \quad \{ p_i : \tau_i \parallel \Gamma_i \}_{i \in L} & \quad \text{by IH, part 2 over (4)} \\
(7) \quad \text{tplp}(\{ i \mapsto p_i \}_{i \in L}) : \text{prod}(\{ i \mapsto \tau_i \}_{i \in L}) \parallel \bigcup_{i \in L} \Gamma_i & \quad \text{by Rule (B.4d) on (6)} \\
\end{align*}

Case (B.12d).

\begin{align*}
(1) \quad \hat{p} = \text{prinjp}[\ell](\hat{p}') & \quad \text{by assumption} \\
(2) \quad p = \text{injp}[\ell](p') & \quad \text{by assumption} \\
(3) \quad \tau = \text{sum}(\{ i \mapsto \tau_i \}_{i \in L}; \ell \mapsto \tau') & \quad \text{by assumption} \\
(4) \quad \hat{p} \leadsto p' : \tau' \parallel \hat{\Delta}; \hat{\Phi}; b \hat{\Gamma} & \quad \text{by assumption} \\
(5) \quad p' : \tau' \parallel \Gamma & \quad \text{by IH, part 2 on (4)} \\
(6) \quad \text{injp}[\ell](p') : \text{sum}(\{ i \mapsto \tau_i \}_{i \in L}; \ell \mapsto \tau') \parallel \Gamma & \quad \text{by Rule (B.4e) on (5)} \\
\end{align*}

Case (B.12e).

\begin{align*}
(1) \quad \hat{p} = \text{splicedp}[m,n;\hat{\tau}] & \quad \text{by assumption} \\
(2) \quad \emptyset \vdash \hat{\Delta}; b \vdash \tau & \quad \text{by assumption} \\
(3) \quad \text{parseUExp}(\text{subseq}(b;m;n)) = \hat{p} & \quad \text{by assumption} \\
(4) \quad \hat{\Delta} \vdash \hat{\Phi} \hat{p} \leadsto p : \tau \parallel \hat{\Gamma} & \quad \text{by assumption} \\
(5) \quad p : \tau \parallel \Gamma & \quad \text{by IH, part 1 on (4)} \\
\end{align*}

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:

\[
\| \hat{\Delta} \vdash \hat{\Phi} \hat{p} \leadsto p : \tau \parallel \hat{\Gamma} \| = \| \hat{p} \| \\
\| \hat{p} \leadsto p : \tau \parallel \hat{\Delta}; \hat{\Phi}; b \hat{\Gamma} \| = \| b \|
\]

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:

\[
\| \hat{\Delta} \vdash \hat{\Phi} \hat{p}' \leadsto (b) \leadsto p : \tau \parallel \hat{\Gamma} \| = \| \hat{p} \| \\
\| \hat{p} \leadsto p : \tau \parallel \hat{\Delta}; \hat{\Phi}; b \hat{\Gamma} \| = \| b \|
\]

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

$$\|\hat{\Delta} \downarrow \Phi, \hat{\rho} \sim \rho : \tau \Downarrow \hat{\Gamma}\| < \|\text{splicedp}[m;n;\tau] \sim \rho : \tau \Downarrow \hat{\Delta}; \Phi; b \hat{\Gamma}\|$$

i.e. by the definitions above,

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

This is established by appeal to Condition B.3.7, which states that subsequences of $b$ are no longer than $b$, and the Condition B.3.3, 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.3.7 and B.3.3, we have that $\|\hat{\rho}\| < \|b\|$ as needed.

### B.5.3 Typed Expression Expansion

**Theorem B.5.4** (Typed Expansion (Strong)).

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

   (b) If $\langle D; \Delta \rangle \langle G; \Gamma \rangle \vdash_{\Psi; \Phi} \hat{\rho} \sim r : \tau \Rightarrow \tau'$ then $\Delta \Gamma \vdash r : \tau \Rightarrow \tau'$.

2. (a) If $\Delta \Gamma \vdash_{\langle D; \Delta_{app} \rangle \langle G; \Gamma_{app} \rangle \Psi; \Phi; b} \hat{\rho} \sim e : \tau$ 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 $\Delta \Gamma \vdash_{\langle D; \Delta_{app} \rangle \langle G; \Gamma_{app} \rangle \Psi; \Phi; b} \hat{\rho} \sim r : \tau \Rightarrow \tau'$ and $\Delta \cap \Delta_{app} = \emptyset$ and dom$(\Gamma) \cap$ dom$(\Gamma_{app}) = \emptyset$ then $\Delta \cup \Delta_{app} \Gamma \cup \Gamma_{app} \vdash r : \tau \Rightarrow \tau'$.

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

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

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

   (1) $\hat{\rho} = \hat{\tau}$ 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)

   (b) **Case (B.6b).**

   (1) $\hat{\rho} = \hat{\tau}' : \tau$ by assumption

   (2) $\hat{\Delta} \vdash \tau \sim \tau$ type by assumption

   (3) $\hat{\Delta} \vdash_{\Psi; \Phi} \hat{\rho}' \sim e : \tau$ by assumption

   (4) $\Delta \Gamma \vdash e : \tau$ by IH, part 1(a) on (3)
Case (B.6c).

1. \( \hat{e} = \text{let val } \hat{x} = \hat{e}_1 \text{ in } \hat{e}_2 \) by assumption
2. \( e = \text{ap}(\text{lam}\{\tau_1\}(x.e_2); e_1) \) by assumption
3. \( \hat{\Delta}, \hat{\xi}, \hat{\Phi} \vdash \hat{e}_1 \leadsto e_1 : \tau_1 \) by assumption
4. \( \hat{\Delta}, \hat{\xi}, \hat{\Phi} \vdash x : \tau_1 \vdash \hat{\psi}, \hat{\Phi} \vdash \hat{e}_2 \leadsto e_2 : \tau \) 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. \( \Delta, \Gamma \vdash \text{lam}\{\tau_1\}(x.e_2) : \text{parr}(\tau_1; \tau) \) by Rule (B.2b) on (6)
8. \( \Delta, \Gamma \vdash \text{ap}(\text{lam}\{\tau_1\}(x.e_2); e_1) : \tau \) 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. \( e = \text{lam}\{\tau_1\}(x.e') \) by assumption
3. \( \tau = \text{parr}(\tau_1; \tau_2) \) by assumption
4. \( \hat{\Delta} \vdash \hat{\tau}_1 \leadsto \tau_1 \text{ type} \) by assumption
5. \( \hat{\Delta}, \hat{\xi}, \hat{\Phi} \vdash x : \tau_1 \vdash \hat{\psi}, \hat{\Phi} \vdash \hat{e}' \leadsto e' : \tau_2 \) by assumption
6. \( \Delta \vdash \tau_1 \text{ type} \) by Lemma B.5.1 on (4)
7. \( \Delta, \Gamma, x : \tau_1 \vdash e' : \tau_2 \) by IH, part 1(a) on (5)
8. \( \Delta, \Gamma \vdash \text{lam}\{\tau_1\}(x.e') : \text{parr}(\tau_1; \tau_2) \) by Rule (B.2b) on (6) and (7)

Case (B.6e).

1. \( \hat{e} = \hat{e}_1(\hat{e}_2) \) by assumption
2. \( e = \text{ap}(e_1; e_2) \) by assumption
3. \( \hat{\Delta}, \hat{\xi}, \hat{\Phi} \vdash \hat{e}_1 \leadsto e_1 : \text{parr}(\tau_2; \tau) \) by assumption
4. \( \hat{\Delta}, \hat{\xi}, \hat{\Phi} \vdash \hat{e}_2 \leadsto e_2 : \tau_2 \) by assumption
5. \( \Delta, \Gamma \vdash e_1 : \text{parr}(\tau_2; \tau) \) by IH, part 1(a) on (3)
6. \( \Delta, \Gamma \vdash e_2 : \tau_2 \) by IH, part 1(a) on (4)
7. \( \Delta, \Gamma \vdash \text{ap}(e_1; e_2) : \tau \) by Rule (B.2c) on (5) and (6)

Case (B.6f) through (B.6m). These cases follow analogously, i.e. we apply Lemma B.5.1 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) \( \hat{e} = \hat{a} \leftarrow \hat{a}' \) (by assumption)

(2) \( \Delta \vdash \tau' \rightsquigarrow \tau' \) type (by assumption)

(3) \( \hat{\Gamma} \vdash [\hat{g}; \hat{\Phi} : \hat{e}_{\text{dep}} \rightsquigarrow e_{\text{dep}} : \tau_{\text{dep}}] \) (by assumption)

(4) \( \emptyset \vdash e_{\text{parse}} : \text{parr} \) (by assumption)

(5) \( \hat{\Delta} : \hat{\Psi} ; \hat{\Phi} \vdash e_{\text{dep}} \rightsquigarrow e_{\text{dep}} : \tau_{\text{dep}} \) (by assumption)

(6) \( \Delta \vdash \tau' \) type (by Lemma B.5.1 to (2))

(7) \( \Delta \vdash \tau_{\text{dep}} \) (by Lemma B.5.1 to (3))

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

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

(10) \( e = \text{ap} \left( \lambda \tau_{\text{dep}} \left( x.e' \right) ; e_{\text{dep}} \right) \) (by assumption)

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

Case (B.6o).

(1) \( \hat{e} = \hat{a} \leftarrow \hat{a}' \) (by assumption)

(2) \( A = A', \hat{a} \rightsquigarrow x \) (by assumption)

(3) \( \Psi = \Psi', x \leftarrow \text{setlm} \) (by assumption)

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

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

(6) \( b \downarrow \text{Body} e_{\text{body}} \) (by assumption)

(7) \( e_{\text{parse}} (e_{\text{body}}) \downarrow \text{inj}[\text{Success}] (e_{\text{proto}}) \) (by assumption)

(8) \( e_{\text{proto}} \uparrow \text{Expr} \hat{e} \) (by assumption)

(9) \( \emptyset \vdash [\hat{\Delta} ; \hat{\Psi} ; \hat{\Phi} ; b : \hat{e} \rightsquigarrow e' : \text{parr} (\tau_{\text{dep}} ; \tau)] \) (by assumption)

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

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

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

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

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

(15) \( \Delta \vdash e : \tau \) (by Rule (B.2c) on (13) and (14))
2. In the following, let \( \hat{\Delta} = (\Delta; \Delta_{app}) \) and \( \hat{\Gamma} = (\Gamma; \Gamma_{app}) \).

(a) Case (B.10a).

\[
\begin{align*}
(1) & \ e = x & \text{by assumption} \\
(2) & \ e = x & \text{by assumption} \\
(3) & \ \Delta = \Gamma', x : \tau & \text{by assumption} \\
(4) & \ \Delta \cup \Delta_{app} \Gamma', x : \tau \vdash x : \tau & \text{by Rule (B.2a)}
\end{align*}
\]
(5) \( \Delta \cup \Delta_{\text{app}} \Gamma', x : \tau \cup \Gamma_{\text{app}} \vdash x : \tau \) by Lemma B.2.2 over \( \Gamma_{\text{app}} \) to (4)

Case (B.10d).

(1) \( \hat{e} = \text{prllam}\{\tau_1\}(x.\hat{e}') \) by assumption
(2) \( e = \text{lam}\{\tau_1\}(x.e') \) by assumption
(3) \( \tau = \text{parr}(\tau_1; \tau_2) \) by assumption
(4) \( \Delta \vdash \hat{\Delta}_{\text{app}}^b \tau_1 \leadsto \tau_1 \) type by assumption
(5) \( \Delta \Gamma, x : \tau_1 \vdash \hat{\Delta}_{\text{app}}^b \Psi_{\text{app}}; \Phi; b \hat{e}' \leadsto e_2 : \tau_2 \) 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.5.2 on (4) and (6)
(11) \( \Delta \cup \Delta_{\text{app}} \Gamma, x : \tau_1 \cup \Gamma_{\text{app}} \vdash e_1 : \tau_2 \) by IH, part 2(a) on (5), (6) and (9)
(12) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}}, x : \tau_1 \vdash e_2 : \tau_2 \) by exchange over \( \Gamma_{\text{app}} \) on (11)
(13) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \vdash \text{lam}\{\tau_1\}(x.\hat{e}') : \text{parr}(\tau_1; \tau_2) \) by Rule (B.2b) on (10) and (12)

Case (B.10e).

(1) \( \hat{e} = \text{prap}(\hat{e}_1; \hat{e}_2) \) by assumption
(2) \( e = \text{ap}(\hat{e}_1; \hat{e}_2) \) by assumption
(3) \( \Delta \Gamma \vdash \hat{\Delta}_{\text{app}}^b \Psi_{\text{app}}; \Phi; b \hat{e}_1 \leadsto e_1 : \text{parr}(\tau_2; \tau) \) by assumption
(4) \( \Delta \Gamma \vdash \hat{\Delta}_{\text{app}}^b \Psi_{\text{app}}; \Phi; b \hat{e}_2 \leadsto e_2 : \tau_2 \) 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) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \vdash e_1 : \text{parr}(\tau_2; \tau) \) 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) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \vdash \text{ap}(\hat{e}_1; \hat{e}_2) : \tau \) by Rule (B.2c) on (7) and (8)

Case (B.10f).

(1) \( \hat{e} = \text{prtllam}(t.\hat{e}') \) by assumption
(2) \( e = \text{tlam}(t.e') \) by assumption
(3) \( \tau = \text{all}(t.\tau') \) 

by assumption

(4) \( \Delta, t \text{ type } \Gamma \vdash \hat{\Delta}_\text{app}; \hat{\Psi}; \hat{\Phi}; b \overset{\hat{e}'}{\rightsquigarrow} e' : \tau' \)

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) \( t \text{ type } \notin \Delta_\text{app} \)

by identification convention

by (5) and (7)

by IH, part 2(a) on (4), (8) and (6)

by exchange over \( \Delta_\text{app} \) on (9)

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.5.2 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).

\begin{align*}
(1) \quad & \hat{e} = \text{splice}d[e][m; n; \hat{\tau}] \\
(2) \quad & \mathcal{E} = \langle \mathcal{D}; \Delta_\text{app} \rangle; \langle \mathcal{G}; \Gamma_\text{app} \rangle; \hat{\Psi}; b \\
(3) \quad & \emptyset \vdash \text{ts}(\mathcal{E}) \overset{\check{\tau}}{\rightsquigarrow} \tau \text{ type} \\
(4) \quad & \text{parseUExp}(\text{subseq}(b; m; n)) = \hat{e} \\
(5) \quad & \hat{\Delta}_\text{app} \hat{\Gamma}_\text{app} \vdash \hat{\check{\tau}} \Rightarrow \hat{\tau} \\
(6) \quad & \Delta \cap \Delta_\text{app} = \emptyset \\
(7) \quad & \text{dom}(\Gamma) \cap \text{dom}(\Gamma_\text{app}) = \emptyset \\
(8) \quad & \Delta_\text{app} \Gamma_\text{app} \vdash e : \tau \\
(9) \quad & \Delta \cup \Delta_\text{app} \Gamma \cup \Gamma_\text{app} \vdash e : \tau \\
\end{align*}

by assumption

by IH, part 1 on (5)

by Lemma B.2.2 over \( \Delta \) and \( \Gamma \) and exchange on (8)

Case (B.10o).

\begin{align*}
(1) \quad & \hat{e} = \text{prmatch}(\hat{e}'; \{\hat{r}_i\}_{1 \leq i \leq n}) \\
(2) \quad & e = \text{match}(e'; \{r_i\}_{1 \leq i \leq n}) \\
(3) \quad & \Delta \Gamma \vdash \hat{\Delta} \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b \overset{\hat{e}'}{\rightsquigarrow} e' : \tau' \\
(4) \quad & \{\Delta \Gamma \vdash \hat{\Delta}_i \hat{\Gamma}_i; \hat{\Psi}_i; \hat{\Phi}_i; b \hat{r}_i \overset{\hat{r}_i}{\rightsquigarrow} r_i : \tau' \Rightarrow \tau\}_{1 \leq i \leq n} \\
(5) \quad & \Delta \cap \Delta_\text{app} = \emptyset \\
\end{align*}

by assumption

by assumption

by assumption

by assumption

by assumption

by assumption
(6) \( \text{dom}(\Gamma) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset \) by assumption
(7) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \vdash e' : \tau' \) by IH, part 2(a) on (3), (5) and (6)
(8) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \vdash r : \tau' \Rightarrow \tau \) by IH, part 2(b) on (4), (5) and (6)
(9) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \vdash \text{match}(e'; \{r_i\}_{1 \leq i \leq n}) : \tau \) by Rule (B.2l) on (7) and (8)

(b) There is only one case.

Case (B.11).

(1) \( \hat{r} = \text{prrule}(p, \hat{e}) \) by assumption
(2) \( r = \text{rule}(p, e) \) by assumption
(3) \( p : \tau \vdash \Gamma' \) by assumption
(4) \( \Delta, \Gamma' \vdash \hat{\Delta}; \hat{\Psi}; \hat{\Phi}; b \hat{e} \leadsto e : \tau' \) by assumption
(5) \( \Delta \cap \Delta_{\text{app}} = \emptyset \) by assumption
(6) \( \text{dom}(\Gamma) \cap \text{dom}(\Gamma') = \emptyset \) by identification convention
(7) \( \text{dom}(\Gamma_{\text{app}}) \cap \text{dom}(\Gamma') = \emptyset \) by identification convention
(8) \( \text{dom}(\Gamma) \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset \) by assumption
(9) \( \text{dom}(\Gamma \cup \Gamma') \cap \text{dom}(\Gamma_{\text{app}}) = \emptyset \) by standard finite set definitions and identities on (6), (7) and (8)
(10) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma' \cup \Gamma_{\text{app}} \vdash e : \tau' \) by IH, part 2(a) on (4), (5) and (9)
(11) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \cup \Gamma' \vdash e : \tau' \) by exchange of \( \Gamma' \) and \( \Gamma_{\text{app}} \) on (10)
(12) \( \Delta \cup \Delta_{\text{app}} \Gamma \cup \Gamma_{\text{app}} \vdash \text{rule}(p, e) : \tau \Rightarrow \tau' \) 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:

\[
\|\hat{\Delta}; \hat{\Gamma} \vdash_{\hat{\Psi}; \hat{\Phi}} \hat{\hat{e}} \leadsto e : \tau\| = \|\hat{\hat{e}}\|
\]

\[
\|\Delta; \Gamma \vdash \hat{\Delta}; \hat{\Psi}; \hat{\Phi}; b \hat{e} \leadsto e : \tau\| = \|b\|
\]

where \( \|b\| \) is the length of \( b \) and \( \|\hat{\hat{e}}\| \) is the sum of the lengths of the seTLM literal bodies in \( \hat{\hat{e}} \), as defined in Sec. B.3.1.
The only case in the proof of part 1 that invokes part 2 is Case (B.60). There, we have that the metric remains stable:

\[ \| \hat{\Delta} ; \hat{\Psi} ; \hat{\Phi} \; \hat{\tau} \hat{\rho} (b) \hat{\tau} : \tau \| \]
\[ = \| \hat{\Delta} \vdash \hat{\Psi} ; \hat{\Phi} ; \hat{\rho} b \hat{\tau} : \tau \| \]
\[ = \| b \| \]

The only case in the proof of part 2 that invokes part 1 is Case (B.10n). There, we have that \( \text{parseUExp}(\text{subseq}(b; m; n)) = \hat{\rho} \) and the IH is applied to the judgement \( \hat{\Delta} ; \hat{\Psi} ; \hat{\Phi} ; \hat{\rho} b \hat{\tau} : \tau \). 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:

\[ \| \hat{\Delta} ; \hat{\Psi} ; \hat{\Phi} \; \hat{\tau} \hat{\rho} (b) \hat{\tau} : \tau \| < \| \Delta ; \Gamma \vdash \hat{\Psi} ; \hat{\Phi} ; b \; \text{splicede}[m; n; \hat{\tau}] \hat{\tau} : \tau \| \]

i.e. by the definitions above,

\[ \| \hat{\rho} \| < \| b \| \]

This is established by appeal to Condition B.3.7, which states that subsequences of \( b \) are no longer than \( b \), and Condition B.3.2, 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 \( \| \hat{\rho} \| < \| b \| \) as needed.

**Theorem B.5.5** (Typed Expression Expansion). If \( \langle \Delta \rangle ; \langle \Gamma \rangle \vdash \hat{\Psi} ; \hat{\Phi} \; \hat{\rho} \hat{\tau} : \tau \) then \( \Delta ; \Gamma \vdash e : \tau \).

**Proof.** This theorem follows immediately from Theorem B.5.4, part 1(a).

**B.5.4 Abstract Reasoning Principles**

**Lemma B.5.6** (Proto-Type Expansion Decomposition). If \( \Delta \vdash \langle \Delta_{\text{app}} \rangle ; b \; \hat{\tau} : \tau \) type where \( \text{seg}(\hat{\tau}) = \{ \text{splicede}[m_i; n_i] \}_{0 \leq i < n} \) then all of the following hold:

1. \( \{ \langle \Delta ; \Delta_{\text{app}} \rangle \vdash \text{parseUTyp}(\text{subseq}(b; m_i; n_i)) \; \hat{\tau}_i : \tau_i \} \}_{0 \leq i < n} \)

2. \( \tau = \{ \tau_i \}_{0 \leq i < n} \) for some \( \tau_i \) and fresh \( \{ t_i \}_{0 \leq i < n} \) (i.e. \( \{ t_i \notin \text{dom}(\Delta) \}_{0 \leq i < n} \) and \( \{ t_i \notin \text{dom}(\Delta_{\text{app}}) \}_{0 \leq i < n} \))

3. \( \text{fv}(\rho') \subset \text{dom}(\Delta) \cup \{ t_i \}_{0 \leq i < n} \)

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

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

1. \( \hat{\tau} = t \) by assumption
(2) \( \tau = t \) by assumption
(3) \( \Delta = \Delta', t \) type by assumption
(4) \( \text{seg}(\tau) = \emptyset \) by definition
(5) \( \text{fv}(t) = \{t\} \) by definition
(6) \( \{t\} \subset \text{dom}(\Delta) \cup \emptyset \) 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) \( \hat{\tau} = \text{prparr}(\hat{\tau}_1; \hat{\tau}_2) \) by assumption
(2) \( \tau = \text{parr}(\tau'_1; \tau'_2) \) by assumption
(3) \( \Delta \vdash \text{T}\_\{\tau_1\} \Rightarrow \tau'_1 \) type by assumption
(4) \( \Delta \vdash \text{T}\_\{\tau_2\} \Rightarrow \tau'_2 \) type by assumption
(5) \( \text{seg}(\hat{\tau}) = \text{seg}(\hat{\tau}_1) \cup \text{seg}(\hat{\tau}_2) \) by definition
(6) \( \text{seg}(\hat{\tau}_1) = \{\text{splicedt}[m_i; n_i]\}_{0 \leq i < n'} \) by definition
(7) \( \text{seg}(\hat{\tau}_2) = \{\text{splicedt}[m_i; n_i]\}_{n' \leq i < n} \) by definition
(8) \( \{\langle D; \Delta_{\text{app}} \rangle \vdash \text{parseUTyp(subseq}(b; m_i; n_i)) \Rightarrow \tau_i \}_{0 \leq i < n'} \) by IH on (3) and (6)
(9) \( \tau'_1 = \{\{\tau_i/t_i\}_{0 \leq i < n'}\} \tau''_1 \) for some \( \tau''_1 \) and fresh \( \{t_i\}_{0 \leq i < n'} \) by IH on (3) and (6)
(10) \( \text{fv}(\tau''_1) \subset \text{dom}(\Delta) \cup \{t_i\}_{0 \leq i < n'} \) by IH on (3) and (6)
(11) \( \{\langle D; \Delta_{\text{app}} \rangle \vdash \text{parseUTyp(subseq}(b; m_i; n_i)) \Rightarrow \tau_i \}_{n' \leq i < n} \) by IH on (4) and (7)
(12) \( \tau'_2 = \{\{\tau_i/t_i\}_{n' \leq i < n}\} \tau''_2 \) for some \( \tau''_2 \) and fresh \( \{t_i\}_{n' \leq i < n} \) by IH on (4) and (7)
(13) \( \text{fv}(\tau''_2) \subset \text{dom}(\Delta) \cup \{t_i\}_{n' \leq i < n} \) 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) \( \text{fv}(\tau''_1) \subset \text{dom}(\Delta) \cup \{t_i\}_{0 \leq i < n} \) by (10) and (14)
(16) \( \text{fv}(\tau''_2) \subset \text{dom}(\Delta) \cup \{t_i\}_{0 \leq i < n} \) 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) \[ \tau'_2 = [\{\tau_i/t_i\}_{0 \leq i < n}] \tau''_2 \] by substitution properties and (12) and (14)

(19) \( \text{parr}(\tau'_1; \tau'_2) = [\{\tau_i/t_i\}_{0 \leq i < n}] \text{parr}(\tau''_1; \tau''_2) \) by substitution and (17) and (18)

(20) \( \text{fv}(\text{parr}(\tau'_1; \tau'_2)) = \text{fv}(\tau''_1) \cup \text{fv}(\tau''_2) \) by definition

(21) \( \text{fv}(\text{parr}(\tau'_1; \tau'_2)) \subset \text{dom}(\Delta) \cup \{t_i\}_{0 \leq i < n} \) by (20) and (15) and (16)

The conclusions hold as follows:

1. (8) ∪ (11)
2. Choosing \( \{t_i\}_{0 \leq i < n} \) and \( \text{parr}(\tau''_1; \tau''_2) \), by (19)
3. (21)

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

Case (B.9g).

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

The conclusions hold as follows:

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

Lemma B.5.7 (Proto-Expression and Proto-Rule Expansion Decomposition).

1. If \( \Delta \Gamma \vdash \langle D; \Delta_{app}; (G; \Gamma_{app}); \Phi; \hat{\Phi}; b \hat{\vdash} e : \tau \text{ where seg}(\hat{\vdash}) = \{\text{splicedt}[m';n'_i]\}_{0 \leq i < n_y} \cup \{\text{splicede}[m_i;n_i; \hat{\tau}_i]\}_{0 \leq i < n_{exp}} \) then all of the following hold:
Proof. By rule induction over Rules (B.10) and Rule (B.11). In the following, let \( \hat{\Delta} = \langle D; \Delta_{app} \rangle \) and \( \hat{\Gamma} = \langle G; \Gamma_{app} \rangle \) and \( \hat{E} = \hat{\Delta}; \hat{\Gamma}; \hat{\Psi}; \hat{\Phi}; b \).

1. Case (B.10a).

\begin{enumerate}
\item \( \hat{\epsilon} = x \) by assumption
\item \( \epsilon = x \) by assumption
\item \( \Gamma = \Gamma', x : \tau \) by assumption
\item \( \text{seg}(x) = \{ \} \) by definition
\item \( \text{fv}(x) = \{ x \} \) by definition
\item \( \text{fv}(x) \subseteq \text{dom}(\Gamma) \) by definition
\item \( \text{fv}(x) \subseteq \text{dom}(\Gamma) \cup \text{dom}(\Delta) \) by (6) and definition of subset
\end{enumerate}

The conclusions hold as follows:

\begin{enumerate}
\item This conclusion holds trivially because \( n_{ty} = 0 \).
\item This conclusion holds trivially because \( n_{exp} = 0 \).
\item This conclusion holds trivially because \( n_{exp} = 0 \).
\item Choose \( x, \emptyset \) and \( \emptyset \).
\end{enumerate}
Case (B.10b) through (B.10m). These cases follow by straightforward inductive argument.

Case (B.10n).

1. \( \hat{e} = splicede[m; n; \hat{\tau}] \) by assumption
2. \( \text{seg}(splicede[m; n; \hat{\tau}]) = \text{seg}(\hat{\tau}) \cup \{splicede[m; n; \hat{\tau}]\} \)
3. \( \text{seg}(\hat{\tau}) = \{splicede[m'; n'; \tau']_{0 \leq i < n_y}\} \) by definition
4. \( \emptyset \vdash \text{ts}(E) \hat{\tau} \leadsto \tau \text{ type} \) by assumption
5. \( \text{parseUExp(subseq}(b; m; n)) = \hat{e} \) by assumption
6. \( \langle D; \Delta_{app} \rangle \langle G; \Gamma_{app} \rangle \vdash \hat{\Psi}; \hat{\Phi}; \hat{e} : \tau \) by assumption
7. \( \{\langle D; \Delta_{app} \rangle \vdash \text{parseUTyp(subseq}(b; m'; n'_{i}) \leadsto \tau'_{i})_{0 \leq i < n_y}\} \) by Lemma B.5.6 on (4) and (3)

8. \( x \notin \text{dom}(\Gamma) \) by identification convention
9. \( x \notin \text{dom}(\Gamma_{app}) \) by identification convention
10. \( x \notin \text{dom}(\Delta) \) by identificaiton convention
11. \( x \notin \text{dom}(\Delta_{app}) \) by identification convention
12. \( e = \[\{\tau'_{i}/t_{i}\}_{0 \leq i < n_y}, e/x\]x \) by definition
13. \( \text{fv}(x) = \{x\} \) by definition
14. \( \text{fv}(x) \subset \text{dom}(\Delta) \cup \text{dom}(\Gamma) \cup \{t_{i}\}_{0 \leq i < n_y} \cup \{x\} \) by definition

The conclusions hold as follows:

(a) (7)
(b) \{ (4) \}
(c) \{ (6) \}
(d) Choosing \( x, \{t_{i}\}_{0 \leq i < n_y} \) and \( \{x\} \), by (8), (9), (10), (11) and (12).
(e) (14)

Case (B.10o).

1. \( \hat{e} = \text{prmatch}(\hat{e}'; \{\hat{r}_{i}\}_{1 \leq i \leq n}) \) by assumption
2. \( e = \text{match}(\tau; e')\{r_{i}\}_{1 \leq i \leq n} \) by assumption
3. \( \Delta \Gamma \vdash E \hat{e} \leadsto e : \tau' \) by assumption
4. \( \{\Delta \Gamma \vdash E \hat{r}_{j} \leadsto r_{j} : \tau' \Rightarrow \tau\}_{1 \leq j \leq n} \) by assumption
5. \( \text{seg}(\text{prmatch}(\hat{e}'; \{\hat{r}_{i}\}_{1 \leq i \leq n})) = \text{seg}(\hat{e}) \cup \bigcup_{0 \leq i < n} \text{seg}(\hat{r}_{i}) \) by definition
(6) \( \text{seg}(\varepsilon') = \{\text{splicede}[m'_i; n'_i]\}_{0 \leq i < n'_y} \cup \{\text{splicede}[m'_i; n'_i; \tau_i]\}_{0 \leq i < n'_{\exp}} \) by definition

(7) \( \{\text{seg}(\tau_j) = \{\text{splicede}[m'_{i,j}; n'_{i,j}]\}_{0 \leq i < n_{y,j}} \cup \{\text{splicede}[m_{i,j}; n_{i,j}; \tau_{i,j}]\}_{0 \leq i < n_{\exp,j}}\}_{0 \leq j < n} \) by definition

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

(9) \( \{\emptyset \vdash (D; \text{app}; b; \tau_i) \leadsto \tau_i \text{ type}\}_{0 \leq i < n'_{\exp}} \) by IH, part 1 on (3) and (6)

(10) \( \{\langle D; \Delta; \text{app} \rangle \langle G; \Gamma; \text{app} \rangle \vdash \text{parseUExp}(\text{subseq}(b; m; n_i)) \leadsto e_i : \tau_i\}_{0 \leq i < n_{\exp}} \) by IH, part 1 on (3) and (6)

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

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

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

(14) \( \{\emptyset \vdash (D; \text{app}; b; \tau_{i,j}) \leadsto \tau_{i,j} \text{ type}\}_{0 \leq i < n_{\exp,j}} \}_{0 \leq j < n} \) by IH, part 2 over (4) and (7)

(15) \( \{\langle D; \Delta; \text{app} \rangle \langle G; \Gamma; \text{app} \rangle \vdash \text{parseUExp}(\text{subseq}(b; m; n_i)) \leadsto e_{i,j} : \tau_{i,j}\}_{0 \leq i < n_{\exp,j}} \}_{0 \leq j < n} \) by IH, part 2 over (4) and (7)

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

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

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

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

(20) \( e' = \{\{\tau'_i / t_i\}_{0 \leq i < n'_{y}} \cup \{\tau_{i,j} / t_{i,j}\}_{0 \leq i < n_{y,j}}, \{e_i / x_i\}_{0 \leq i < n_{\exp}} \cup \{0 \leq j < n\} \)
\[ \{ \tau_{ij}/t_{ij} \}_{0 \leq i < n_{ty}} \] by substitution properties and (11) and (12) and (18) and (19)

(21) \[ \{ r_j = [\{ \tau_{ij}^j'/t_{ij} \}_{0 \leq i < n_{ty}'} \cup \{ \tau_{ij}/t_{ij} \}_{0 \leq i < n_{ty}} ]_{0 \leq j < n} \{ e_i/x_i \}_{0 \leq i < n_{exp}} \cup \{ 0 \leq j < n \} \] by substitution properties and (16) and (17) and (18) and (19)

(22) \[ e = [\{ \tau_{ij}^j'/t_{ij} \}_{0 \leq i < n_{ty}'} \cup \{ \tau_{ij}/t_{ij} \}_{0 \leq i < n_{ty}} ]_{0 \leq j < n} \{ e_i/x_i \}_{0 \leq i < n_{exp}} \cup \{ 0 \leq j < n \} \text{match}(e'', \{ r_i' \}_{1 \leq i \leq n}) \] by (20) and (21) and definition of substitution

(23) \[ \text{fv}(e'') \subseteq \text{dom}(\Delta) \cup \text{dom}(\Gamma) \cup \{ t_i \}_{0 \leq i < n_{ty}'} \cup \{ t_{ij} \}_{0 \leq i < n_{ty}} \cup \{ x_i \}_{0 \leq i < n_{exp}} \cup \{ x_{ij} \}_{0 \leq i < n_{exp}, 0 \leq j < n} \] by (12) and (18) and (19)

(24) \[ \{ \text{fv}(r_i' \}_{1 \leq i \leq n} \} \subseteq \text{dom}(\Delta) \cup \text{dom}(\Gamma) \cup \{ t_i \}_{0 \leq i < n_{ty}'} \cup \{ t_{ij} \}_{0 \leq i < n_{ty}} \cup \{ x_i \}_{0 \leq i < n_{exp}} \cup \{ x_{ij} \}_{0 \leq i < n_{exp}, 0 \leq j < n} \] by (17) and (18) and (19)

(25) \[ \text{fv} (\text{match}(e''; \{ r_i' \}_{1 \leq i \leq n})) \subseteq \text{dom}(\Delta) \cup \text{dom}(\Gamma) \cup \{ t_i \}_{0 \leq i < n_{ty}'} \cup \{ t_{ij} \}_{0 \leq i < n_{ty}} \cup \{ x_i \}_{0 \leq i < n_{exp}} \cup \{ x_{ij} \}_{0 \leq i < n_{exp}, 0 \leq j < n} \text{match}(e'', \{ r_i' \}_{1 \leq i \leq n}) \] by (23) and (24)

The conclusions hold as follows:

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

We have \[ e = [\{ \tau_{ij}^j'/t_{ij} \}_{0 \leq i < n_{ty}'} \cup \{ \tau_{ij}/t_{ij} \}_{0 \leq i < n_{ty}} ]_{0 \leq j < n} \{ e_i/x_i \}_{0 \leq i < n_{exp}} \cup \{ e_i/x_{ij} \}_{0 \leq i < n_{exp}, 0 \leq j < n} \text{match}(e''; \{ r_i' \}_{1 \leq i \leq n}) \text{ by (22).}

(e) (25)

2. By rule induction over the rule typing assumption. There is only one case. In the following, let \( \hat{\Lambda} = (\mathcal{D}; \Delta_{app}) \) and \( \hat{\Gamma} = (\mathcal{G}; \Gamma_{app}) \) and \( \mathbb{E} = \hat{\Lambda}; \hat{\Gamma}; \Psi; \Phi; b. \)
Theorem B.5.8 (seTLM Abstract Reasoning Principles). If \( \langle D; \Delta \rangle \langle G; \Gamma \rangle \vdash_{\Psi, \Phi} \hat{a} \; \hat{b} \vdash \hat{e} : \tau \) then:

1. (Expansion Typing) \( \hat{\Psi} = \Psi', \hat{a} \vdash x \mapsto \text{set} \in \tau \) and \( \Delta \Gamma \vdash e : \tau \)
2. (Responsibility) \( b \downarrow_{\text{Body}} e_{\text{body}} \) and \( e_{\text{parse}}(e_{\text{body}}) \downarrow \text{inj}([\text{SuccessE}] (e_{\text{proto}}) \) and \( e_{\text{proto}} \uparrow_{\text{PrExpr}} \hat{e} \)

3. (Segmentation) \( \text{seg}(\hat{e}) \) segments \( b \)

4. (Segment Typing) \( \text{seg}(\hat{e}) = \{\text{spliced}[m_i'; n_i']\}_{0 \leq i < n_{ty}} \cup \{\text{spliced}[m_i; n_{ti}; \hat{\tau}_i]\}_{0 \leq i < n_{exp}} \) and
   
   (a) \( \langle D; \Delta \rangle \vdash \text{parseUTyp}((\text{subseq}(b; m_i'; n_i')) \leadsto \tau'_i \) type \( 0 \leq i < n_{ty} \) and \( \{\Delta \vdash \tau'_i \) type \( 0 \leq i < n_{ty} \)
   
   (b) \( \emptyset \vdash \langle D; \Delta \rangle; b \hat{\tau}_i \leadsto \tau_i \) type \( 0 \leq i < n_{exp} \) and \( \{\Delta \vdash \tau_i \) type \( 0 \leq i < n_{exp} \)
   
   (c) \( \langle D; \Delta \rangle \langle G; \Gamma \rangle \vdash \Phi \vdash \text{parseUExp}((\text{subseq}(b; m_i; n_i)) \leadsto e_i : \tau_i)_{0 \leq i < n_{exp}} \) and \( \{\Delta \vdash \tau'_i \leadsto e_i : \tau_i\}_{0 \leq i < n_{exp}} \)

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

6. (Context Independence) \( \text{fv}(e') \subset \{\tau_i\}_{0 \leq i < n_{ty}} \cup \{x_i\}_{0 \leq i < n_{exp}} \)

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

Case (B.6o).

   (1) \( \breve{\Phi} = \breve{\Phi}', \hat{x} \leadsto x \leftrightarrow \text{setlm}(\tau; e_{\text{parse}}) \) by assumption
   
   (2) \( \Gamma = \Gamma', x : \tau_{\text{dep}} \) by assumption
   
   (3) \( e = \text{ap}(e_{\text{parse}}; x) \) by assumption
   
   (4) \( \langle D; \Delta \rangle \langle G; \Gamma \rangle \vdash \Phi \vdash \hat{\tau} \leadsto (\hat{\tau}' \leadsto e : \tau) \) by assumption
   
   (5) \( \Delta \vdash e : \tau \) by Theorem B.5.5 on (4)
   
   (6) \( b \downarrow_{\text{Body}} e_{\text{body}} \) by assumption
   
   (7) \( e_{\text{parse}}(e_{\text{body}}) \downarrow \text{inj}([\text{SuccessE}] (e_{\text{proto}}) \) by assumption
   
   (8) \( e_{\text{proto}} \uparrow_{\text{PrExpr}} \hat{e} \) by assumption
   
   (9) \( \text{seg}(\hat{e}) \) segments \( b \) by assumption
   
   (10) \( \emptyset \vdash \breve{\Delta}_f; \Phi; b \hat{\epsilon} \leadsto e_x : \text{parr}(\tau_{\text{dep}}; \tau) \) by assumption
   
   (11) \( \text{seg}(\hat{e}) = \{\text{spliced}[m_i'; n_i']\}_{0 \leq i < n_{ty}} \cup \{\text{spliced}[m_i; n_{ti}; \hat{\tau}_i]\}_{0 \leq i < n_{exp}} \) by definition
   
   (12) \( \{\langle D; \Delta \rangle \vdash \text{parseUTyp}((\text{subseq}(b; m_i'; n_i')) \leadsto \tau'_i \) type \( 0 \leq i < n_{ty} \) \) by Lemma B.5.7 on (10) and (11)
   
   (13) \( \{\Delta \vdash \tau'_i \) type \( 0 \leq i < n_{ty} \) \) by Lemma B.5.1, part 1 over (12)

49
\[ \{ \emptyset \vdash \langle D; \Delta ; b \rangle ; \tau_i \leadsto \tau_i \text{ type} \} \subseteq \{ \emptyset \vdash \tau_i \text{ type} \} \subseteq b \tau_i \vdash \tau_i \text{ type} \subseteq \{ \emptyset \vdash \langle D ; \Delta \rangle ; b \hat{\Gamma} \vdash \hat{\Phi} \} \] 0 \leq i < n_{\text{exp}} \\

by Lemma B.5.7 on (10) and (11)

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

(16) \{ \Delta \vdash \tau_i \text{ type} \} \subseteq \{ \emptyset \vdash \tau_i \text{ type} \} \\
by Lemma B.5.1, part 2 over (14) and (15)

(17) \{ \langle D ; \Delta \rangle ; \langle G ; \Gamma \rangle \vdash \hat{\Psi} ; \hat{\Phi} \} \subseteq \{ \emptyset \vdash \langle D ; \Delta \rangle \hat{\Gamma} \vdash \hat{\Phi} \} \\
by Lemma B.5.7 on (10) and (11)

(18) \{ \Delta \Gamma \vdash e_i : \tau_i \} \subseteq \{ \emptyset \vdash \tau_i \text{ type} \} \\
by Theorem B.5.5 over (17)

(19) e_x = [\{ \tau'_i / t_i \} \subseteq \{ e_i / x_i \} ] \\
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.5.9** (Proto-Pattern Expansion Decomposition). If \( \hat{\Phi} \vdash e : \tau \vdash \hat{\Phi} \) where

\[
\text{seg}(\hat{\Phi}) = \{ \text{spliceda}[m'_i; n'_i] \} \subseteq \{ \text{spliced}[m_i; n_i] \}
\]

then all of the following hold:

1. \( \{ \hat{\Delta} \vdash \text{parseUTyp}(\text{subseq}(b; m'_i; n'_i)) \leadsto \tau'_i \text{ type} \} \subseteq \{ \emptyset \vdash \hat{\Phi} \} \)
2. \( \{ \emptyset \vdash \hat{\Phi} \} \subseteq \{ \hat{\Delta} \vdash \tau_i \leadsto \tau_i \text{ type} \} \)

\[ \]  

50
3. \( \{ \hat{\Delta} \vdash \Phi \text{ parseUPat}(\text{subseq}(b; m_i; n_i)) \leadsto p_i : \tau_i \vdash \hat{\Gamma}_i \}_{0 \leq i < n_{\text{pat}}} \)

4. \( \hat{\Gamma} = \bigcup_{0 \leq i < n_{\text{pat}}} \hat{\Gamma}_i \)

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

**Case (B.12a).**

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

The conclusions hold as follows:
1. This conclusion holds trivially because \( n_{\text{ty}} = 0 \).
2. This conclusion holds trivially because \( n_{\text{pat}} = 0 \).
3. This conclusion holds trivially because \( n_{\text{pat}} = 0 \).
4. This conclusion holds trivially because \( \hat{\Gamma} = \emptyset \) and \( n_{\text{pat}} = 0 \).

**Case (B.12b).**

1. \( \hat{p} = \text{prfoldp}(\hat{p}') \) by assumption
2. \( p = \text{foldp}(p') \) by assumption
3. \( \tau = \text{rec}(t.\tau') \) by assumption
4. \( \hat{p} \leadsto p : [\text{rec}(t.\tau') / t] \tau' \vdash \hat{P} \hat{\Gamma} \) by assumption
5. \( \text{seg(prfoldp}(\hat{p}') = \text{seg}(\hat{p}') \) by definition
6. \( \text{seg}(\hat{p}') = \{\text{spliced}(m_i; n_i')\}_{0 \leq i < n_{\text{ty}}} \cup \{\text{spliced}(m_i; n_i; \hat{\tau}_i)\}_{0 \leq i < n_{\text{pat}}} \) by definition
7. \( \{ \hat{\Delta} \vdash \text{parseUTyp}(\text{subseq}(b; m_i'; n_i')) \leadsto \hat{\tau}_i \text{ type} \}_{0 \leq i < n_{\text{ty}}} \) by IH on (4) and (6)
8. \( \{ \emptyset \vdash \hat{\Delta}; b \hat{\tau}_i \leadsto \hat{\tau}_i \text{ type} \}_{0 \leq i < n_{\text{pat}}} \) by IH on (4) and (6)
9. \( \{ \hat{\Delta} \vdash \Phi \text{ parseUPat}(\text{subseq}(b; m_i; n_i)) \leadsto p_i : \tau_i \vdash \hat{\Gamma}_i \}_{0 \leq i < n_{\text{pat}}} \) by IH on (4) and (6)
10. \( \hat{\Gamma} = \bigcup_{0 \leq i < n_{\text{pat}}} \hat{\Gamma}_i \) by IH on (4) and (6)

The conclusions hold as follows:
1. (7)
Case (B.12c).

1. \( \hat{p} = \text{prtplp}[L](\{j \mapsto \hat{p}_j\}_{j \in L}) \) by assumption
2. \( p = \text{tplp}(\{j \mapsto p_j\}_{j \in L}) \) by assumption
3. \( \tau = \text{prod}(\{j \mapsto \tau_j\}_{j \in L}) \) by assumption
4. \( \hat{\Gamma} = \bigcup_{j \in L} \hat{\Gamma}_j \) by assumption
5. \( \{p_j \equiv p_j : \tau_j \vdash P \hat{\Gamma}_j\}_{j \in L} \) by assumption
6. \( \text{seg}(\text{prtplp}[L](\{j \mapsto \hat{p}_j\}_{j \in L})) = \bigcup_{j \in L} \text{seg}(\hat{p}_j) \) by definition
7. \( \{\text{seg}(\hat{p}_j) = \{\text{spliced}[m'_{i,j}; n'_{i,j}]\}_{0 \leq i < n_{\text{ty},j}} \cup \{\text{splicedp}[m_{i,j}; n_{i,j}; \hat{\tau}_{ij}]\}_{0 \leq i < n_{\text{pat},j}}\}_{j \in L} \) by definition
8. \( n_{\text{pat}} = \sum_{j \in L} n_{\text{pat},j} \) by definition
9. \( \{\hat{\Delta} \vdash \text{parseUTyp}(\text{subseq}(b; m'_{i,j}; n'_{i,j})) \equiv \tau'_{ij}, \text{type}\}_{0 \leq i < n_{\text{ty},j}}\}_{j \in L} \) by IH over (5) and (7)
10. \( \{\emptyset \vdash \hat{\Delta}; b \hat{\tau}_{ij} \equiv \tau_{ij}, \text{type}\}_{0 \leq i < n_{\text{pat},j}}\}_{j \in L} \) by IH over (5) and (7)
11. \( \{\hat{\Delta} \vdash \Phi \text{parseUPat}(\text{subseq}(b; m_{i,j}; n_{i,j})) \equiv p_{i,j} : \tau_{ij} \vdash P \hat{\Gamma}_{ij}\}_{0 \leq i < n_{\text{pat},j}}\}_{j \in L} \) by IH over (5) and (7)
12. \( \hat{\Gamma}_j = \bigcup_{0 \leq i < n_{\text{pat},j}} \hat{\Gamma}_{ij}\}_{j \in L} \) by IH over (5) and (7)
13. \( \bigcup_{j \in L} \hat{\Gamma}_j = \bigcup_{j \in L} \bigcup_{i < n_{\text{pat},j}} \hat{\Gamma}_{ij}\) by definition and (12)

The conclusions hold as follows:

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

Case (B.12d).

1. \( \hat{p} = \text{prinjp}[\ell](\hat{p}') \) by assumption
2. \( p = \text{injp}[\ell](p') \) by assumption
3. \( \tau = \text{sum}(\{i \mapsto \tau_i\}_{i \in L}; \ell \mapsto \tau') \) by assumption
(4) \( \hat{p} \leadsto p : \tau' \parallel^P \hat{\Gamma} \) by assumption

(5) \( \text{seg(prinjp}[\ell](\hat{p}')) = \text{seg}(\hat{p}') \) by definition

(6) \( \text{seg}(\hat{p}') = \{\text{splicedp}[m_i'; n_i']\}_{0 \leq i < n_{ty}} \cup \{\text{splicedp}[m_i; n_i; \tau_i]\}_{0 \leq i < n_{pat}} \) by definition

(7) \{\( \hat{\Delta} \vdash \text{parseUTyp}(\text{subseq}(b; m_i'; n_i')) \leadsto \tau'_i\) type\}_{0 \leq i < n_{ty}} by IH on (4) and (6)

(8) \{\( \emptyset \vdash \hat{\Delta}; \hat{\tau}_i \leadsto \tau_i\) type\}_{0 \leq i < n_{pat}} by IH on (4) and (6)

(9) \{\( \hat{\Delta} \vdash \hat{\Phi}, \text{parseUPat}(\text{subseq}(b; m_i; n_i)) \leadsto p_i : \tau_i \parallel \hat{\Gamma}_i\)\}_{0 \leq i < n_{pat}} by IH on (4) and (6)

(10) \( \hat{\Gamma} = \bigcup_{0 \leq i < n_{pat}} \hat{\Gamma}_i \) by IH on (4) and (6)

The conclusions hold as follows:

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

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

(1) \( \hat{p} = \text{splicedp}[m; n; \hat{\tau}] \) by assumption

(2) \( \emptyset \vdash \hat{\Delta}; \hat{\tau} \leadsto \tau\) type by assumption

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

(4) \( \hat{\Delta} \vdash \hat{\Phi}, \hat{p} \leadsto p : \tau \parallel \hat{\Gamma} \) by assumption

(5) \( \text{seg}(\text{splicedp}[m; n; \hat{\tau}]) = \text{seg}(\hat{\tau}) \cup \{\text{splicedp}[m; n; \hat{\tau}]\} \) by definition

(6) \( \text{seg}(\hat{\tau}) = \{\text{splicedd}[m_i'; n_i']\}_{0 \leq i < n_{ty}} \) by definition

(7) \{\( \langle D; \Delta_{app} \rangle \vdash \text{parseUTyp}(\text{subseq}(b; m_i; n_i)) \leadsto \tau_i\) type\}_{0 \leq i < n} by Lemma B.5.6 on (2) and (6)

The conclusions hold as follows:

1. (7)
2. (2)
3. (3) and (4)
4. This conclusion holds by (4) because \( n_{pat} = 1 \).
Theorem B.5.10 (spTLM Abstract Reasoning Principles). If $\hat{\Delta} \vdash _\Phi \hat{a} \, \langle \text{b} \rangle \, \leadsto \, p : \tau \vdash \hat{\Gamma}$ 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)** $\hat{\Phi} = \hat{\Phi}', \hat{a} \leadsto x \leftrightarrow \text{sptlm}(\tau; e_{\text{parse}})$ and $p : \tau \vdash \hat{\Gamma}$
2. **(Responsibility)** $b \downarrow_{\text{Body}} e_{\text{body}}$ and $e_{\text{parse}}(e_{\text{body}}) \downarrow \text{inj}[\text{SuccessP}] (e_{\text{proto}})$ and $e_{\text{proto}} \uparrow_{\text{PrPat}} p$
3. **(Segmentation)** $\text{seg}(\hat{p})$ segments $b$
4. **(Segment Typing)** $\text{seg}(\hat{p}) = \{\text{splicd}[m'_i;n'_i]\}_{0 \leq i < n_{ty}} \cup \{\text{splicd}[m_i;n_i;\hat{\tau}_i]\}_{0 \leq i < n_{pat}}$ and
   
   (a) $\{\hat{\Delta} \vdash \text{parseUTyp}(\text{subseq}(b;m'_i;n'_i)) \leadsto \hat{\tau}'_i \text{type}\}_{0 \leq i < n_{ty}}$ and $\{\Delta \vdash \hat{\tau}'_i \text{type}\}_{0 \leq i < n_{ty}}$
   
   (b) $\{\emptyset \vdash \hat{\Delta}; b \; \hat{\tau}_i \leadsto \hat{\tau}_i \text{type}\}_{0 \leq i < n_{pat}}$ and $\{\Delta \vdash \hat{\tau}_i \text{type}\}_{0 \leq i < n_{pat}}$
   
   (c) $\hat{\Delta} \vdash _\Phi \text{parseUPat}(\text{subseq}(b;m_i;n_i)) \leadsto p_i : \hat{\tau}_i \vdash \langle \mathcal{G}_i; \Gamma_i \rangle$ and $\{p_i : \hat{\tau}_i -\vdash \langle \mathcal{G}_i; \Gamma_i \rangle\}_{0 \leq i < n_{pat}}$
5. **(Visibility)** $\mathcal{G} = \bigcup_{0 \leq i < n_{pat}} \mathcal{G}_i$ 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. $\hat{\Delta} \vdash _\Phi \hat{a} \, \langle \text{b} \rangle \, \leadsto \, p : \tau \vdash \hat{\Gamma}$ by assumption
2. $\hat{\Phi} = \hat{\Phi}', \hat{a} \leadsto x \leftrightarrow \text{sptlm}(\tau; e_{\text{parse}})$ by assumption
3. $p : \tau \vdash \hat{\Gamma}$ by Theorem B.5.3 on (1)
4. $b \downarrow_{\text{Body}} e_{\text{body}}$ by assumption
5. $e_{\text{parse}}(e_{\text{body}}) \downarrow \text{inj}[\text{SuccessP}] (e_{\text{proto}})$ by assumption
6. $e_{\text{proto}} \uparrow_{\text{PrPat}} p$ by assumption
7. $\text{seg}(\hat{p})$ segments $b$ by assumption
8. $\hat{p} \leadsto p : \tau \vdash \hat{\Delta}; \hat{\Phi}; b \; \hat{\Gamma}$ by assumption
9. $\text{seg}(\hat{p}) = \{\text{splicd}[m'_i;n'_i]\}_{0 \leq i < n_{ty}} \cup \{\text{splicd}[m_i;n_i;\hat{\tau}_i]\}_{0 \leq i < n_{pat}}$ by definition
10. $\{\hat{\Delta} \vdash \text{parseUTyp}(\text{subseq}(b;m'_i;n'_i)) \leadsto \hat{\tau}'_i \text{type}\}_{0 \leq i < n_{ty}}$ by Lemma B.5.9 on (8) and (9)
11. $\{\Delta \vdash \hat{\tau}'_i \text{type}\}_{0 \leq i < n_{ty}}$ by Lemma B.5.1, part 1 over (10)
(12) \( \{ \emptyset \vdash \hat{\Delta}; \hat{\tau}_i \rightsquigarrow \tau_i \text{ type} \}_{0 \leq i < n_{\text{pat}}} \) \hspace{1cm} by Lemma B.5.9 on (8) and (9)

(13) \( \{ \Delta \vdash \tau_i \text{ type} \}_{0 \leq i < n_{\text{pat}}} \) \hspace{1cm} by Lemma B.5.1, part 2 over (12)

(14) \( \{ \hat{\Delta} \vdash \hat{\Phi}_{\text{parseUPat}}(\text{subseq}(b; m_i; n_i)) \rightsquigarrow p_i : \tau_i \models \hat{\Gamma}_i \}_{0 \leq i < n_{\text{pat}}} \) \hspace{1cm} by Lemma B.5.9 on (8) and (9)

(15) \( \{ p_i : \tau_i \models \Gamma_i \}_{0 \leq i < n_{\text{pat}}} \) \hspace{1cm} by Theorem B.5.3 over (14)

(16) \( G = \bigcup_{0 \leq i < n_{\text{pat}}} G_i \) and \( \Gamma = \bigcup_{0 \leq i < n_{\text{pat}}} \Gamma_i \) \hspace{1cm} by Lemma B.5.9 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