A Translation Proof of Nominal Wyvern Soundness

Yu Xiang Zhu[*] **Julian Mackay**[†] **Alex Potanin**[†]
Jonathan Aldrich[*]

October 2019
CMU-ISR-19-102

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Institute for Software Research, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[†]Victoria University of Wellington, Wellington, New Zealand

Abstract

This technical report proves type safety for Nominal Wyvern with a translation to pDOT, a DOT-based system with general paths that has been proven to be type safe.

Keywords: Type Safety, Wyvern

1 Introduction

Nominal Wyvern [4] is a new core type system for Wyvern [1] based on the DOT calculus [2]. It achieves a higher degree of nominality in a DOT-based system by semantically separating the definition of structures and their subtype relations from arbitrary width refinements and type bounds. This contributes to a system with more explicit meanings and relations, useful for both human readers to reason about and programming tools to refer to. At the same time, nominality helps achieve subtype decidability. In line with the theme of semantic separation, Nominal Wyvern adapts material-shape separation so that decidability results from an intuitive separation of types with different roles. This contributes to a restriction that is more easily understandable and articulable. The resulting system preserves the ability to express common patterns expressible with DOT, at the same time allowing for patterns that will be familiar to programmers already used to traditional functional or object-oriented programming languages.

This technical report proves type safety for a slightly updated version of the grammar. Section 2 presents the syntax, static semantics, and dynamic semantics of an updated version of Nominal Wyvern. Section 3 proves type safety with a type preserving translation to pDOT. Section 4 concludes the report.

2 Grammar

The version of Nominal Wyvern in this technical report has been updated from the thesis version in [4]. The main difference lies in the semantics of refinements.

2.1 Refinements as Intersections

In the thesis version of Nominal Wyvern, refinements were treated as modifications to the base type it refines. Formally, the type bounds in refinements were required to be no wider than the corresponding bounds in the base type. However, this restriction is stricter than what is needed, and is not always possible to check, especially after environment narrowing during reduction. Therefore, this restriction is relaxed so that the type bound in a refinement does not have to be related to the corresponding bound in the base type it refines.

As a result, a type is now treated as an intersection of its base type with its refinement. In fact, a base type is allowed to be followed by more than one refinement during subtyping. We call a type with multiple refinements an “extended type”. An extended type is considered an intersection of its base type with all of its refinements.

2.2 Syntax

Figure 1 presents the syntax of the updated Nominal Wyvern. It is unchanged from the thesis version except for the removal of the `if` expression. It was removed for conciseness since it is not central to the purpose of this language design.

$$
\begin{array}{lll}
P & ::= & \textbf{\textit{program:}} \\
 & \overline{D}\ e & \\[1em]
D & ::= & \textbf{\textit{top-level decls:}} \\
 & \mathbf{name}\ n\ \{x \Rightarrow \overline{\sigma}\} & \textit{named type decl} \\
 & \mathbf{subtype}\ n\, r <: n & \textit{subtype decl} \\[1em]
B & ::= & \textbf{\textit{type bound:}} \\
 & \leq & \textit{upper bound} \\
 & \geq & \textit{lower bound} \\
 & = & \textit{exact bound} \\[1em]
\sigma & ::= & \textbf{\textit{member decl:}} \\
 & \mathbf{type}\ t\ B\ \tau & \textit{type member decl} \\
 & \mathbf{val}\ v\ :\ \tau & \textit{field decl} \\
 & \mathbf{def}\ f\ :\ \tau\ x \rightarrow \tau & \textit{method decl} \\[1em]
r & ::= & \textbf{\textit{refinement:}} \\
 & \{\overline{\delta}\} & \\[1em]
\delta & ::= & \textbf{\textit{refinement member decl:}} \\
 & \mathbf{type}\ t\ B\ \tau &
\end{array}
$$

$$
\begin{array}{lll}
\beta & ::= & \textbf{\textit{base type:}} \\
 & \bot & \textit{bottom type} \\
 & \top & \textit{top type} \\
 & n & \textit{named type} \\
 & p.t & \textit{path type} \\[1em]
\tau & ::= & \textbf{\textit{type:}} \\
 & \beta\ r & \\[1em]
p & ::= & \textbf{\textit{path:}} \\
 & x & \textit{variable} \\
 & l & \textit{store location*} \\
 & p.v & \textit{val selection} \\[1em]
e & ::= & \textbf{\textit{expression:}} \\
 & p & \textit{path} \\
 & p.f(p) & \textit{method application} \\
 & \mathbf{new}\ \tau\{x \Rightarrow \overline{d}\} & \textit{new object} \\
 & \mathbf{let}\ x = e\ \mathbf{in}\ e & \textit{let expr} \\[1em]
d & ::= & \textbf{\textit{object member defn:}} \\
 & \mathbf{type}\ t\ =\ \tau & \textit{type member defn} \\
 & \mathbf{val}\ v\ :\ \tau_v\ =\ e & \textit{field defn} \\
 & \mathbf{def}\ f\ :\ \tau\ x \rightarrow \tau = e & \textit{method defn}
\end{array}
$$

$$
\begin{array}{lll}
\Delta ::= & \overline{n : \{x \Rightarrow \overline{\sigma}\}} & \textit{Name Definition Context} \\
\Sigma ::= & \overline{n\ r <: n} & \textit{Name Subtype Relation Context} \\
\Gamma ::= & \overline{x : \tau} & \textit{Variable Typing Context} \\
S ::= & \overline{l : \tau} & \textit{Location Typing Context} \\
\mu ::= & \overline{l : \{x \Rightarrow \overline{d}\}} & \textit{Runtime Store}
\end{array}
$$

* Intermediate form only (not user accessible)

Figure 1: Nominal Wyvern Syntax

2.3 Static Semantics

2.3.1 Top Level Well-Formedness

$\boxed{P : \tau}$

$$\frac{\text{names}(\overline{D}) = \Delta \qquad \forall\, n : \{x \Rightarrow \overline{\sigma}\} \in \Delta.\ \Delta\Sigma \vdash n : \{x \Rightarrow \overline{\sigma}\}\ \text{wf} \\ \text{subs}(\overline{D}) = \Sigma \qquad \forall\, n_1\, r_1 <:\ n_2 \in \Sigma.\ \Delta\Sigma \vdash n_1\, r_1 <:\ n_2 \qquad \Delta\Sigma\cdot\cdot \vdash e : \tau}{\overline{D}\, e : \tau}\ \textsc{TL-WF}$$

$\boxed{\Delta\Sigma \vdash n : \{x \Rightarrow \overline{\sigma}\}\ \text{wf}}$

$$\frac{\forall\, \texttt{type}\ t\ B\ \tau_t \in \overline{\sigma}.\ \forall\, \texttt{val}\ v\ :\ \tau_v \in \overline{\sigma}.\ v \notin \tau_t \text{ and } \Delta\Sigma(x : n)\cdot \vdash \tau_t\ \text{wf}}{\Delta\Sigma \vdash n : \{x \Rightarrow \overline{\sigma}\}\ \text{wf}}\ \textsc{NAME-WF}$$

$\boxed{\Delta\Sigma \vdash n_1\, r_1 <:\ n_2}$

$$\frac{\Delta(n_1) = \{x_1 \Rightarrow \overline{\sigma_1}\} \\ \Delta(n_2) = \{x_2 \Rightarrow \overline{\sigma_2}\} \qquad r_1 = \{\overline{\delta_1}\} \qquad \Delta\Sigma(x_1 : n_1\, r_1)\cdot \vdash \overline{\sigma_1}, \overline{\delta_1} <:\ [x_1/x_2]\overline{\sigma_2}}{\Delta\Sigma \vdash n_1\, r_1 <:\ n_2}\ \textsc{SUB-WF}$$

$\boxed{\Delta\Sigma\Gamma S \vdash \tau\ \text{wf}}$

$$\frac{}{\Delta\Sigma\Gamma S \vdash \top\ \text{wf}}\ \textsc{TYPE-WF-TOP} \qquad \frac{}{\Delta\Sigma\Gamma S \vdash \bot\, \overline{r}\ \text{wf}}\ \textsc{TYPE-WF-BOT}$$

$$\frac{\Delta\Sigma\Gamma S \vdash \beta\, r_\beta \prec n\, r_n \qquad \Delta(n) = \{x_n \Rightarrow \overline{\sigma_n}\} \\ \forall\, \texttt{type}\ t\ B\ \tau \in \overline{r}.\ \Delta\Sigma\Gamma S \vdash n\, r_n \ni_x \texttt{type}\ t\ B'\ \tau'}{\Delta\Sigma\Gamma S \vdash \beta\, \overline{r}\ \text{wf}}\ \textsc{TYPE-WF}$$

names($\overline{D}$) $= \Delta$ creates a names definition context Δ from $\overline{D}$ by matching each name to its definition verbatim.
subs($\overline{D}$) $= \Sigma$ creates a name subtype relation context Σ by copying over each subtype declaration in $\overline{D}$ verbatim.

Figure 2: Nominal Wyvern Top Level Well-Formedness

The top level declarations at the start of each Nominal Wyvern program defines all the named types and subtype relations between them. Figures 2 and 3 present the rules that judge the well-formedness of top level declarations. The changes from the thesis version are

1. The merge operation on $\overline{\delta}$, $+_\delta$, is replaced with concatenation (in line with the philosophy of refinements as intersections).

$\boxed{\Delta\Sigma\Gamma S \vdash \overline{\sigma} <: \overline{\sigma}}$

$$\frac{}{\Delta\Sigma\Gamma S \vdash \overline{\sigma} <: \cdot}\ \textsc{SDL-Emp} \qquad \frac{\begin{array}{c} r_1 \ni \texttt{\textbf{type}}\ t\ B_1\ \tau_1 \\ \Delta\Sigma\Gamma S \vdash \texttt{\textbf{type}}\ t\ B_1\ \tau_1 <: \texttt{\textbf{type}}\ t\ B_2\ \tau_2 \\ \Delta\Sigma\Gamma S \vdash \overline{\sigma_1} <: \overline{\sigma_2} \end{array}}{\Delta\Sigma\Gamma S \vdash \overline{\sigma_1} <: \overline{\sigma_2}, \texttt{\textbf{type}}\ t\ B_2\ \tau_2}\ \textsc{SDL-Type}$$

$$\frac{\begin{array}{c} r_1 \ni \texttt{\textbf{val}}\ v\ :\ \tau_1 \\ \Delta\Sigma\Gamma S \vdash \texttt{\textbf{val}}\ v\ :\ \tau_1 <: \texttt{\textbf{val}}\ v\ :\ \tau_2 \\ \Delta\Sigma\Gamma S \vdash \overline{\sigma_1} <: \overline{\sigma_2} \end{array}}{\Delta\Sigma\Gamma S \vdash \overline{\sigma_1} <: \overline{\sigma_2}, \texttt{\textbf{val}}\ v\ :\ \tau_2}\ \textsc{SDL-Val}$$

$$\frac{\begin{array}{c} r_1 \ni \texttt{\textbf{def}}\ f\ :\ \tau_{a1}\ x_1 \to \tau_{r1} \\ \Delta\Sigma\Gamma S \vdash \texttt{\textbf{def}}\ f\ :\ \tau_{a1}\ x_1 \to \tau_{r1} <: \texttt{\textbf{def}}\ f\ :\ \tau_{a2}\ x_2 \to \tau_{r2} \\ \Delta\Sigma\Gamma S \vdash \overline{\sigma_1} <: \overline{\sigma_2} \end{array}}{\Delta\Sigma\Gamma S \vdash \overline{\sigma_1} <: \overline{\sigma_2}, \texttt{\textbf{def}}\ f\ :\ \tau_{a2}\ x_2 \to \tau_{r2}}\ \textsc{SDL-Def}$$

$\Delta\Sigma\Gamma S \vdash \sigma <: \sigma$ *is the subtype judgment on* σ*s (defined later).*

Figure 3: Nominal Wyvern Top Level Well-Formedness (continued)

2. The type well-formedness rules no longer require bounds in refinements to be no wider than the corresponding bounds in the base type. The only requirement is that refinements do not introduce new members not present in the base type.

2.3.2 Term Typing

Figure 4 presents the term typing rules of Nominal Wyvern. The main changes from the thesis version are

1. Removed typing rule for `if` expressions as they were removed from the grammar.
2. A new well-formedness rule for `new` expressions. In particular, the exposed type of the object is required to be non-bottom (`NB`). Non-bottomness ensures that the type is never narrowed into $\perp$ in the future by statically fixing the named type (and thus the structure) of the new object. This is needed to ensure type preservation.

2.3.3 Subtyping

Figures 5 and 6 presents the updated subtyping rules. As mentioned in Section 2.1, refinements are allowed to stack together during subtype checking (replacing the earlier behavior of discarding older refinements upon conflict). Due to the possibility of multiple refinements to the same type member, the member access rule is now non-deterministic in which refinement (or the base type) to return the bounds from.

$\boxed{\Delta\Sigma\Gamma S \vdash e : \tau}$

$$\frac{\Gamma(x) = \tau}{\Delta\Sigma\Gamma S \vdash x : \tau}\ \text{T-VAR} \qquad \frac{S(l) = \tau}{\Delta\Sigma\Gamma S \vdash l : \tau}\ \text{T-LOC} \qquad \frac{\Delta\Sigma\Gamma S \vdash p : \tau \quad \Delta\Sigma\Gamma S \vdash \tau \prec n\,\overline{r} \quad \Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \texttt{val}\ v\ :\ \tau_v}{\Delta\Sigma\Gamma S \vdash p.v : [p/x]\tau_v}\ \text{T-SEL}$$

$$\frac{\Delta\Sigma\Gamma S \vdash p : \tau \quad \Delta\Sigma\Gamma S \vdash \tau \prec n\,\overline{r} \quad \Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \texttt{def}\ f\ :\ \tau_a\ x_a \to \tau_r \quad \Delta\Sigma\Gamma S \vdash p' : \tau' \quad \Delta\Sigma\Gamma S \vdash \tau' <:\ [p/x]\tau_a}{\Delta\Sigma\Gamma S \vdash p.f(p') : [p, p_a/x, x_a]\tau_r}\ \text{T-APP} \qquad \frac{\Delta\Sigma\Gamma S \vdash \tau\ \text{wf} \quad \Delta\Sigma\Gamma S \vdash \tau\{x \Rightarrow \overline{d}\}\ \text{wf}}{\Delta\Sigma\Gamma S \vdash \texttt{new}\ \tau\{x \Rightarrow \overline{d}\} : \tau}\ \text{T-NEW}$$

$$\frac{\Delta\Sigma\Gamma S \vdash e_x : \tau_x' \quad \Delta\Sigma\Gamma S \vdash \tau_x' <:\ \tau_x \quad \Delta\Sigma\Gamma, x : \tau_x\, S \vdash e : \tau' \quad \Delta\Sigma\Gamma, x : \tau_x S \vdash \tau' <:\ \tau \quad x \notin fv(\tau)}{\Delta\Sigma\Gamma S \vdash \texttt{let}\ x = e_x\ \texttt{in}\ e : \tau}\ \text{T-LET}$$

$\boxed{\Delta\Sigma\Gamma S \vdash \tau\,\{x \Rightarrow \overline{d}\}\ \text{wf}}$

$$\frac{\begin{array}{c}\Delta\Sigma\Gamma S \vdash \tau\ \text{NB} \quad \Delta\Sigma\Gamma S \vdash \tau \prec n\,\overline{r} \quad \Delta(n) = \{x_n \Rightarrow \overline{\sigma_n}\} \quad \tau_x = n\,\text{ref}(\text{sig}(\overline{d}))\\ \Gamma' = \Gamma, x : \tau_x \quad \Delta\Sigma\Gamma S \vdash \text{sig}(\overline{d}) <:\ \overline{\sigma_n}, \overline{r}\\ \forall\, \texttt{val}\ v\ :\ \tau_v\ =\ p \in \overline{d}.\ \Delta\Sigma\Gamma S \vdash p : \tau_v' \wedge \Delta\Sigma\Gamma S \vdash \tau_v' <:\ \tau_v\\ \forall\, \texttt{def}\ f\ :\ \tau_a\ x_a \to \tau_r = e \in \overline{d}.\ \Delta\Sigma\Gamma, x : \tau_x, x_a : \tau_a\, S \vdash e : \tau_r' \wedge \Delta\Sigma\Gamma, x : \tau_x, x_a : \tau_a S \vdash \tau_r' <:\ \tau_r\end{array}}{\Delta\Sigma\Gamma S \vdash \tau\,\{x \Rightarrow \overline{d}\}\ \text{wf}}$$

$\boxed{\Delta\Sigma\Gamma S \vdash \tau\ \text{NB}}$

$$\frac{}{\Delta\Sigma\Gamma S \vdash \top\ \text{NB}}\ \text{NB-TOP} \qquad \frac{}{\Delta\Sigma\Gamma S \vdash n\,r\ \text{NB}}\ \text{NB-NAME}$$

$$\frac{\Delta\Sigma\Gamma S \vdash p : \tau_p \quad \Delta\Sigma\Gamma S \vdash \tau_p \prec n_p\,\overline{r_p} \quad \Delta\Sigma\Gamma S \vdash n_p\,\overline{r_p} \ni_x \texttt{type}\ t\ =\ \beta_t\,r_t \quad \Delta\Sigma\Gamma S \vdash \beta_t\ \text{NB}}{\Delta\Sigma\Gamma S \vdash p.t\,r\ \text{NB}}\ \text{NB-PATH}$$

$x \notin fv(\tau)$ is true if x is not a free variable in τ.
sig : $\overline{d} \to \overline{\sigma}$ transforms object member definitions into member declarations by removing the dynamic expression part of fields and methods.
ref : $\overline{\sigma} \to r$ filters member declarations by preserving only type member declarations.
$\Delta\Sigma\Gamma S \vdash \tau \prec \tau_u$ is true if following the upper bound of τ leads to τ_u, whose upper bound is itself. Judgments not formally defined here are defined with the subtyping rules.

Figure 4: Nominal Wyvern Term Typing

$\boxed{\Delta\Sigma\Gamma S \vdash \beta\,\overline{r} <:\ \beta\,\overline{r}}$

$$\frac{}{\Delta\Sigma\Gamma S \vdash n\,\overline{r} <:\ \top}\ \textsc{S-Top} \qquad \frac{}{\Delta\Sigma\Gamma S \vdash \bot <:\ n\,\overline{r}}\ \textsc{S-Bot}$$

$$\frac{\begin{array}{c}\Delta\Sigma\Gamma S \vdash p : \tau_p \qquad \Delta\Sigma\Gamma S \vdash \tau_p \prec n\,\overline{r} \qquad \Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \texttt{type}\ t \leqq \beta_t\, r_t \\ \Delta\Sigma\Gamma S \vdash [p/x](\beta_t\, r_t, \overline{r_1}) <:\ \beta_2\, \overline{r_2}\end{array}}{\Delta\Sigma\Gamma S \vdash p.t\,\overline{r_1} <:\ \beta_2\,\overline{r_2}}\ \textsc{S-Upper}$$

$$\frac{\begin{array}{c}\Delta\Sigma\Gamma S \vdash p : \tau_p \qquad \Delta\Sigma\Gamma S \vdash \tau_p \prec n\,\overline{r} \qquad \Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \texttt{type}\ t \geqq \beta_t\, r_t \\ \Delta\Sigma\Gamma S \vdash \beta_1\,\overline{r_1} <:\ [p/x](\beta_t\, r_t, \overline{r_2})\end{array}}{\Delta\Sigma\Gamma S \vdash \beta_1\,\overline{r_1} <:\ p.t\,\overline{r_2}}\ \textsc{S-Lower}$$

$$\frac{\Delta\Sigma\Gamma S \vdash \overline{r_1} <:\ \overline{r_2}}{\Delta\Sigma\Gamma S \vdash \beta\,\overline{r_1} <:\ \beta\,\overline{r_2}}\ \textsc{S-Refl}$$

$$\frac{\Delta\Sigma\Gamma S \vdash n_1 \xrightarrow{\overline{r_1}} n_2 \qquad n1 \neq n2 \qquad \Delta\Sigma\Gamma S \vdash \overline{r_1} <:\ \overline{r_2}}{\Delta\Sigma\Gamma S \vdash n_1\,\overline{r_1} <:\ n_2\,\overline{r_2}}\ \textsc{S-Name}$$

$\boxed{\Delta\Sigma\Gamma S \vdash \overline{r} <:\ \overline{r}}$

$$\frac{\begin{array}{c}\forall\delta_2\,.\,\Delta\Sigma\Gamma S \vdash \overline{r_2} \ni \delta_2 \\ \exists\delta_1\ s.t.\ \Delta\Sigma\Gamma S \vdash \overline{r_1} \ni \delta_1 \wedge \Delta\Sigma\Gamma S \vdash \delta_1 <:\ \delta_2\end{array}}{\Delta\Sigma\Gamma S \vdash \overline{r_1} <:\ \overline{r_2}}\ \textsc{SR-List}$$

$\boxed{\Delta\Sigma\Gamma S \vdash \beta\,\overline{r} \prec \beta\,\overline{r}}$

$$\frac{}{\Delta\Sigma\Gamma S \vdash \top \prec \top}\ \textsc{TE-Top} \qquad \frac{}{\Delta\Sigma\Gamma S \vdash \bot\,\overline{r} \prec \bot\,\overline{r}}\ \textsc{TE-Bot} \qquad \frac{}{\Delta\Sigma\Gamma S \vdash n\,\overline{r} \prec n\,\overline{r}}\ \textsc{TE-Name}$$

$$\frac{\begin{array}{c}\Delta\Sigma\Gamma S \vdash p : \tau_p \qquad \Delta\Sigma\Gamma S \vdash \tau_p \prec n\,\overline{r} \\ \Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \texttt{type}\ t \leqq \beta_t\, r_t \qquad \Delta\Sigma\Gamma S \vdash [p/x](\beta_t\, r_t, \overline{r_1}) \prec \beta_2\,\overline{r_2}\end{array}}{\Delta\Sigma\Gamma S \vdash p.t\,\overline{r_1} \prec \beta_2\,\overline{r_2}}\ \textsc{TE-Upper}$$

$$\frac{\Delta\Sigma\Gamma S \vdash p : \tau_p \qquad \Delta\Sigma\Gamma S \vdash \tau_p \prec n\,\overline{r} \qquad \Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \texttt{type}\ t\ \geq\ \tau_t}{\Delta\Sigma\Gamma S \vdash p.t\,\overline{r_1} \prec p.t\,\overline{r_1}}\ \textsc{TE-Lower}$$

where $\texttt{type}\ t \,{}^{B_1}_{B_2}\, \tau$ *matches a type member declaration with either bound* B_1 *or* B_2*.*

Figure 5: Nominal Wyvern Subtyping

$$\boxed{\Delta\Sigma\Gamma S \vdash n \xrightarrow{\overline{r}} n}$$

$$\frac{}{\Delta\Sigma\Gamma S \vdash n \xrightarrow{\overline{r}} n} \text{ SN-REFL}$$

$$\frac{\Sigma \ni n_1\, r_1 <:\ n_2 \qquad \Delta\Sigma\Gamma S \vdash \overline{r'_1} <:\ r_1 \qquad \Delta\Sigma\Gamma S \vdash n_2 \xrightarrow{\overline{r'_1}} n_3}{\Delta\Sigma\Gamma S \vdash n_1 \xrightarrow{\overline{r'_1}} n_3} \text{ SN-TRANS}$$

$$\boxed{\Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni \delta}$$

$$\frac{\Delta(n) = \{x \Rightarrow \overline{\sigma_n}\} \qquad \Delta\Sigma\Gamma S \vdash \mathrm{ref}(\mathrm{sig}(\overline{\sigma_n})), \overline{r} \ni \delta}{\Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \delta} \text{ M-NAME}$$

$$\boxed{\Delta\Sigma\Gamma S \vdash \overline{r} \ni \delta}$$

$$\frac{\delta \in \overline{\delta}}{\Delta\Sigma\Gamma S \vdash \{\overline{\delta}\} \ni \delta} \text{ M-ND}$$

$$\boxed{\Delta\Sigma\Gamma S \vdash \sigma <:\ \sigma}$$

$$\frac{\Delta\Sigma\Gamma S \vdash \tau_1 <:\ \tau_2}{\Delta\Sigma\Gamma S \vdash \mathbf{type}\ t \leqq \tau_1 <:\ \mathbf{type}\ t\ \leq\ \tau_2} \text{ SS-UPPER}$$

$$\frac{\Delta\Sigma\Gamma S \vdash \tau_2 <:\ \tau_1}{\Delta\Sigma\Gamma S \vdash \mathbf{type}\ t \geqq \tau_1 <:\ \mathbf{type}\ t\ \geq\ \tau_2} \text{ SS-LOWER}$$

$$\frac{\Delta\Sigma\Gamma S \vdash \tau_1 <:\ \tau_2 \qquad \Delta\Sigma\Gamma S \vdash \tau_2 <:\ \tau_1}{\Delta\Sigma\Gamma S \vdash \mathbf{type}\ t\ =\ \tau_1 <:\ \mathbf{type}\ t\ =\ \tau_2} \text{ SS-EXACT}$$

$$\frac{\Delta\Sigma\Gamma S \vdash \tau_1 <:\ \tau_2}{\Delta\Sigma\Gamma S \vdash \mathbf{val}\ v\ :\ \tau_1 <:\ \mathbf{val}\ v\ :\ \tau_2} \text{ SS-VAL}$$

$$\frac{\Delta\Sigma\Gamma S \vdash \tau_{a2} <:\ \tau_{a1} \qquad \Delta\Sigma\Gamma, x_1 : \tau_{a2}\, S \vdash \tau_{r1} <:\ [x_1/x_2]\tau_{r2}}{\Delta\Sigma\Gamma S \vdash \mathbf{def}\ f\ :\ \tau_{a1}\, x_1 \to \tau_{r1} <:\ \mathbf{def}\ f\ :\ \tau_{a2}\, x_2 \to \tau_{r2}} \text{ SS-DEF}$$

$\sigma \in \overline{\delta}$ is true if σ is a type member declaration (i.e. δ), and is part of $\overline{\delta}$.
$x \notin \Gamma$ is true when x is a fresh variable under the current variable typing context.

Figure 6: Nominal Wyvern Subtyping (continued)

$\boxed{\mu \mid e \longmapsto \mu \mid e}$

$$\frac{\mu \vdash p \ \rightarrow \ l \qquad \mu(l) \ni_{x_s} \texttt{def}\ f \ : \ \tau_x\ x \rightarrow \tau_r = e_f}{\mu \mid p.f(p_a) \ \longmapsto \ \mu' \mid [p/x_s, p_a/x]e_f} \text{ EV-APP}$$

$$\frac{}{\mu \mid \texttt{let}\ x = p_1\ \texttt{in}\ e_2 \ \longmapsto \ \mu \mid [p_1/x]e_2} \text{ EV-LET-PATH}$$

$$\frac{l \text{ fresh in } \mu}{\mu \mid \texttt{let}\ x = \texttt{new}\ \tau\{x_s \Rightarrow \overline{d}\}\ \texttt{in}\ e_2 \ \longmapsto \ \mu, l : \{x_s \Rightarrow \overline{d}\} \mid [l/x]e_2} \text{ EV-LET-NEW}$$

$$\frac{e_1 \text{ not a path or new expression} \qquad \mu \mid e_1 \ \longmapsto \ \mu' \mid e_1'}{\mu \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \ \longmapsto \ \mu' \mid \texttt{let}\ x = e_1'\ \texttt{in}\ e_2} \text{ EV-LET}$$

$\boxed{\mu \vdash p \ \rightarrow \ l}$

$$\frac{l \in \mu}{\mu \vdash l \ \rightarrow \ l} \text{ EVP-LOC}$$

$$\frac{\mu \vdash p \ \rightarrow \ l \qquad \mu(l) \ni_{x_s} \texttt{val}\ v \ : \ \tau_v \ = \ p_v \qquad \mu \vdash [p/x_s]p_v \ \rightarrow \ l_v}{\mu \vdash p.v \ \rightarrow \ l_v} \text{ EVP-PATH}$$

where $\mu(l) \ni_x d$ is true if the definition d exists in the definition list stored at memory location l of μ, with the self variable denoted by x,
and $\Delta\Sigma\Gamma S \vdash \mu$ is true if $\forall l : \{x \Rightarrow \overline{d}\} \in \mu\ \exists l : \tau \in S$ s.t. $\Delta\Sigma \cdot S \vdash \tau\{x \Rightarrow \overline{d}\}$ wf.

Figure 7: Nominal Wyvern Reduction Rules

2.4 Dynamic Semantics

Figure 7 presents the small step dynamic semantics of Nominal Wyvern. The "runtime" includes a heap storage μ that stores memory locations. Each memory location l contains the definition of an object created via `new`.

The evaluation rules follow the straightforward interpretation of the expressions. Memory locations represent real objects during runtime, and are treated as values: evaluation stops when the entire program reduces into a location. Paths are evaluated by evaluating field accesses from the root of the path to the leaf. Method applications are evaluated by first evaluating the path to the object containing the method, and then evaluating the argument. Finally replacing all mentions of the argument (and self variable) in the method body with the actual argument location (and the method-containing object's own location). `New` expressions create a new memory location in μ and evaluate immediately to that location. Finally, `let` expressions evaluate the inner expression into a value before substituting it into the outer expression.

3 Type Safety Proof

In proving type safety for Nominal Wyvern, we turn to existing type safe languages. In particular, we look at pDOT [3], a DOT-based system that supports general paths (recall that DOT systems did not used to support paths that are not just a variable). This translation target was chosen due to its similarity to Nominal Wyvern in being related to DOT and supporting general-length paths.

Since pDOT is already proven to be type safe, soundness for Nominal Wyvern can be proven by showing that a translation from Nominal Wyvern programs into pDOT programs exist, that a well-typed program in Nominal Wyvern is well-typed in pDOT after this translation, and that reduction steps taken in Nominal Wyvern correspond to the same reductions taken by the translation in pDOT.

Figure 8 presents a translation from Nominal Wyvern programs into pDOT programs. Since both languages are based on DOT and support general paths, the main job of a translation lies in converting nominal types and their subtyping relations. All named types are encoded into a pDOT object available at the start of the program called `TL`. Each named type becomes a type member of `TL`, bounded on both sides by its named type definition. More effort is required to encode subtype relations since Nominal Wyvern verifies nominal subtyping relations assuming all nominal subtyping relations already hold. This means not all subtyping relations that can be verified in Nominal Wyvern hold naturally in pDOT. To make this work, the translation takes advantage of intersection types by intersecting onto the definition of each named type n all other named types that are declared supertypes of n. Consequently, the concept of conditional subtyping is dropped here. Refinements are broken into individual type declarations that intersect together with the base type. Methods are translated into fields containing lambda objects. All other language constructs map directly into the same construct in pDOT.

Note that the translated program allows more subtype relations to be true than those in Nominal Wyvern due to the relaxation of nominal subtype restrictions (i.e. a program that may not typecheck in Nominal Wyvern may typecheck in pDOT after translation). However, this does not influence the evaluation of the program once it is typechecked in Nominal Wyvern first.

We first wish to prove that the translation is type preserving.

To simplify notation, assume for the rest of this section that the top level variable produced by the translation for a program $\overline{D}\,e$ is already in the pDOT context $\Gamma_{\overline{D}} = TL : v(TL : [\![\overline{D}]\!]^{T}_{\Delta\Sigma})[\![\overline{D}]\!]^{t}_{\Delta\Sigma}$.

Lemma 1 (Translation Preserves Typing)**.** *Given well-formed top-level $\overline{D}$, if $\Delta\Sigma\Gamma S \vdash e : \tau$, then $\Gamma_{\overline{D}}, [\![\Gamma]\!]_{\Delta}, [\![S]\!]_{\Delta} \vdash [\![e]\!]_{\Delta} : [\![\tau]\!]_{\Delta}$.*

Proof. To simplify notation, denote $\Gamma_{\overline{D}}, [\![\Gamma]\!]_{\Delta}, [\![S]\!]_{\Delta}$ with Γ', and $[\![e]\!]_{\Delta}$ with t'.

Assume the premise $\Delta\Sigma\Gamma S \vdash e : \tau$ (1). Prove by induction on term typing:

- *Case* T-VAR:

  Let $e = x$.

  By the translation rules, $t' = x$, and Γ' contains $x : [\![\tau]\!]_{\Delta}$.

  By VAR, $\Gamma' \vdash t' : [\![\tau]\!]_{\Delta}$

$$
\begin{array}{rlcl}
\text{Top Level} & & & \\
\boxed{P \rightsquigarrow t} & [\![\overline{D}\, e]\!]_{\Delta\Sigma} & = & \texttt{let}\ TL = \nu(TL : [\![\overline{D}]\!]^{T}_{\Delta\Sigma})[\![\overline{D}]\!]^{t}_{\Delta\Sigma} \\
 & & & \quad \texttt{in}\ [\![e]\!]_{\Delta} \\
\boxed{D \rightsquigarrow T} & [\![\texttt{name}\ n\ \{x \Rightarrow \overline{\sigma}\}]\!]^{T}_{\Delta\Sigma} & = & \{n : T_n ... T_n\} \\
\boxed{D \rightsquigarrow t} & [\![\texttt{name}\ n\ \{x \Rightarrow \overline{\sigma}\}]\!]^{t}_{\Delta\Sigma} & = & \{n = T_n\} \\
 & & & \textit{where}\ T_n = \mu(x : [\![\overline{\sigma}]\!]_{\Delta}) \wedge T_{\text{sub}} \\
 & & & \textit{and}\ T_{\text{sub}} = \bigwedge_{\texttt{subtype}\ n\, r <: n' \in \Sigma} TL.n' \\
\text{Types} & & & \\
\boxed{\sigma \rightsquigarrow T} & [\![\texttt{val}\ v\ :\ \tau]\!]_{\Delta} & = & \{v : [\![\tau]\!]_{\Delta}\} \\
 & [\![\texttt{def}\ f\ :\ \tau_1\ x \to \tau_2]\!]_{\Delta} & = & \{f : \forall(x : [\![\tau_1]\!]_{\Delta})[\![\tau_2]\!]_{\Delta}\} \\
 & [\![\texttt{type}\ t\ \leq\ \tau]\!]_{\Delta} & = & \{t : [\![\tau]\!]_{\Delta} ... \top\} \\
 & [\![\texttt{type}\ t\ \geq\ \tau]\!]_{\Delta} & = & \{t : \bot ... [\![\tau]\!]_{\Delta}\} \\
 & [\![\texttt{type}\ t\ =\ \tau]\!]_{\Delta} & = & \{t : [\![\tau]\!]_{\Delta} ... [\![\tau]\!]_{\Delta}\} \\
\boxed{\beta \rightsquigarrow T} & [\![\top]\!]_{\Delta} & = & \top \\
 & [\![\bot]\!]_{\Delta} & = & \bot \\
 & [\![p.t]\!]_{\Delta} & = & [\![p]\!]_{\Delta}.t \\
 & [\![n]\!]_{\Delta} & = & TL.n \\
\boxed{\tau \rightsquigarrow T} & [\![\beta\{\overline{\delta}\}]\!]_{\Delta} & = & [\![\beta]\!]_{\Delta} \wedge [\![\overline{\delta}]\!]_{\Delta} \\
\text{Terms} & & & \\
\boxed{e \rightsquigarrow t} & [\![x]\!]_{\Delta} & = & x \\
 & [\![l]\!]_{\Delta} & = & x_l \\
 & [\![p.v]\!]_{\Delta} & = & [\![p]\!]_{\Delta}.v \\
 & [\![p_1.f(p_2)]\!]_{\Delta} & = & [\![p_1]\!]_{\Delta}.f\ [\![p_2]\!]_{\Delta} \\
 & [\![\texttt{new}\ \tau\{x \Rightarrow \overline{d}\}]\!]_{\Delta} & = & [\![\{x \Rightarrow \overline{d}\}]\!]_{\Delta} \\
 & [\![\texttt{let}\ x = e_1\ \texttt{in}\ e_2]\!]_{\Delta} & = & \texttt{let}\ x = [\![e_1]\!]_{\Delta}\ \texttt{in}\ [\![e_2]\!]_{\Delta} \\
\boxed{d \rightsquigarrow d} & [\![\texttt{type}\ t\ =\ \tau]\!]_{\Delta} & = & \{t = [\![\tau]\!]_{\Delta}\} \\
 & [\![\texttt{val}\ v\ :\ \tau\ =\ p]\!]_{\Delta} & = & \{v = [\![p]\!]_{\Delta}\} \\
 & [\![\texttt{def}\ f\ :\ \tau_1\ x \to \tau_2 = e]\!]_{\Delta} & = & \{f = \lambda(x : [\![\tau_1]\!]_{\Delta})[\![e]\!]_{\Delta}\} \\
\text{Contexts} & & & \\
\boxed{\Gamma \rightsquigarrow \Gamma} & [\![\emptyset]\!]_{\Delta} & = & \emptyset \\
 & [\![\Gamma, x : \tau]\!]_{\Delta} & = & [\![\Gamma]\!]_{\Delta}, x : [\![\tau]\!]_{\Delta} \\
\boxed{S \rightsquigarrow \Gamma\ \text{(inert)}} & [\![\emptyset]\!]_{\Delta} & = & \emptyset \\
 & [\![S, l : \tau]\!]_{\Delta} & = & [\![S]\!]_{\Delta}, x_l : [\![\tau]\!]_{\Delta} \\
\boxed{\mu \rightsquigarrow \gamma} & [\![\emptyset]\!]_{\Delta} & = & \emptyset \\
 & [\![\mu, l : \{x \Rightarrow \overline{d}\}]\!]_{\Delta} & = & [\![\mu]\!]_{\Delta}, x_l \mapsto [\![\{x \Rightarrow \overline{d}\}]\!]_{\Delta} \\
\boxed{\{x \Rightarrow \overline{d}\} \rightsquigarrow v} & [\![\{x \Rightarrow \overline{d}\}]\!]_{\Delta} & = & \nu(x : [\![sig(\overline{d})]\!]_{\Delta})[\![\overline{d}]\!]_{\Delta}
\end{array}
$$

The translation of any list of construct is the intersection of the translation of each element: $[\![\overline{s}]\!] = \bigwedge_{s \in \overline{s}} [\![s]\!]$ *for all meta-variables* s.

Figure 8: Nominal Wyvern to pDOT translation

- *Case* T-LOC:

  Similar to Case T-VAR.

- *Case* T-SEL:

  Let $e = p.v$.

  By the translation rules, $t' = [\![p]\!]_\Delta.v$.

  (1) implies:

$$\begin{aligned}
&\Delta\Sigma\Gamma S \vdash p : \tau_p && (2)\\
&\Delta\Sigma\Gamma S \vdash \tau_p \prec n\,\overline{r} && (3)\\
&\Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \texttt{val}\ v\ :\ \tau_v && (4)\\
&\text{and } \tau = [p/x]\tau_v && (5)
\end{aligned}$$

  By IH,

$$\begin{aligned}
(2) &\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : [\![\tau_p]\!]_\Delta \\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : [\![n\,\overline{r}]\!]_\Delta && \text{[(3), expansion pres.]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : \mu(x : \{v : [\![\tau_v]\!]_\Delta\}) && \text{[(4), access pres.]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : [[\![p]\!]_\Delta/x]\{v : [\![\tau_v]\!]_\Delta\}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : \{v : [[\![p]\!]_\Delta/x][\![\tau_v]\!]_\Delta\}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta.v : [[\![p]\!]_\Delta/x][\![\tau_v]\!]_\Delta && \text{[FLD-E]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta.v : [\![[p/x]\tau_v]\!]_\Delta && \text{[substitution pres.]}\\
&\Rightarrow \Gamma' \vdash [\![e]\!]_\Delta : [\![\tau]\!]_\Delta && \text{[(5)]}
\end{aligned}$$

- *Case* T-APP:

  Let $e = p_1.f(p_2)$.

  By the translation rules, $t' = p_1.f\ p_2$.

  (1) implies:

$$\begin{aligned}
&\Delta\Sigma\Gamma S \vdash p_1 : \tau_1 && (2)\\
&\Delta\Sigma\Gamma S \vdash \tau_1 \prec n_1\,\overline{r_1} && (3)\\
&\Delta\Sigma\Gamma S \vdash n_1\,\overline{r_1} \ni_x \texttt{def}\ f\ :\ \tau_a\ x_a \to \tau_r && (4)\\
&\Delta\Sigma\Gamma S \vdash p_2 : \tau_2 && (5)\\
&\Delta\Sigma\Gamma S \vdash \tau_2 <:\ [p_1/x]\tau_a && (6)\\
&\text{and } \tau = [p_1, p_2/x, x_a]\tau_r && (7)
\end{aligned}$$

By IH,

$$\begin{aligned}
(2) &\Rightarrow \Gamma' \vdash [\![p_1]\!]_\Delta : [\![\tau_1]\!]_\Delta \\
&\Rightarrow \Gamma' \vdash [\![p_1]\!]_\Delta : [\![n_1\, \overline{r_1}]\!]_\Delta && \text{[(3), expansion pres.]} \\
&\Rightarrow \Gamma' \vdash [\![p_1]\!]_\Delta : \mu(x : \{f : \forall(x_a : [\![\tau_a]\!]_\Delta)[\![\tau_r]\!]_\Delta\}) && \text{[(4), access pres.]} \\
&\Rightarrow \Gamma' \vdash [\![p_1]\!]_\Delta : [[\![p_1]\!]_\Delta/x]\{f : \forall(x_a : [\![\tau_a]\!]_\Delta)[\![\tau_r]\!]_\Delta\} \\
&\Rightarrow \Gamma' \vdash [\![p_1]\!]_\Delta.f : [[\![p_1]\!]_\Delta/x]\forall(x_a : [\![\tau_a]\!]_\Delta)[\![\tau_r]\!]_\Delta \\
&\Rightarrow \Gamma' \vdash [\![p_1]\!]_\Delta.f : \forall(x_a : [[\![p_1]\!]_\Delta/x][\![\tau_a]\!]_\Delta)[[\![p_1]\!]_\Delta/x][\![\tau_r]\!]_\Delta
\end{aligned}$$

By IH,

$$\begin{aligned}
(5) &\Rightarrow \Gamma' \vdash [\![p_2]\!]_\Delta : [\![\tau_2]\!]_\Delta \\
(6) &\Rightarrow \Gamma' \vdash [\![\tau_2]\!]_\Delta <: [[\![p_1]\!]_\Delta/x][\![\tau_a]\!]_\Delta && \text{[subtype pres.]} \\
&\Rightarrow \Gamma' \vdash [\![p_2]\!]_\Delta : [[\![p_1]\!]_\Delta/x][\![\tau_a]\!]_\Delta && [\textsc{Sub}] \\
&\Rightarrow \Gamma' \vdash [\![p_1]\!]_\Delta.f[\![p_2]\!]_\Delta : [[\![p_2]\!]_\Delta/x_a][[\![p_1]\!]_\Delta/x][\![\tau_r]\!]_\Delta \\
&\Rightarrow \Gamma' \vdash [\![e]\!]_\Delta : [\![\tau]\!]_\Delta
\end{aligned}$$

- *Case* T-NEW:

  Let $e = \textbf{new}\ \tau\{x \Rightarrow \overline{d}\}$.

  By the translation rules, $t' = v(x : [\![\tau]\!]_\Delta)[\![\overline{d}]\!]_\Delta$.

  (1) implies:

$$\begin{aligned}
&\Delta\Sigma\Gamma S \vdash \tau \text{ wf} && (2) \\
&\Delta\Sigma\Gamma S \vdash \tau\{x \Rightarrow \overline{d}\} \text{ wf} && (3)
\end{aligned}$$

  Case on the type of member:

  - *For vals*:

$$\begin{aligned}
(3)\forall \textbf{val}\ v\ :\ \tau_v\ =\ p_v \in \overline{d}. \\
&\Delta\Sigma\Gamma S \vdash n \ni_{x'} \textbf{val}\ v\ :\ \tau'_v && (4) \\
&\Delta\Sigma\Gamma S \vdash p_v : \tau_{va} && (5) \\
&\Delta\Sigma\Gamma S \vdash \tau_{va} <: \tau_v \\
&\Delta\Sigma\Gamma, x' : \tau_x S \vdash \tau_v <: \tau'_v
\end{aligned}$$

    By IH,

$$\begin{aligned}
(5) &\Rightarrow \Gamma' \vdash [\![p_v]\!]_\Delta : [\![\tau_{va}]\!]_\Delta \\
&\Rightarrow \Gamma' \vdash [\![p_v]\!]_\Delta : [\![\tau_v]\!]_\Delta && \text{[subtype pres.]} \\
&\Rightarrow \Gamma', x : [\![\tau_x]\!]_\Delta \vdash [\![p_v]\!]_\Delta : [\![[x/x']\tau_v]\!]_\Delta && \text{[weakening]} \\
&\Rightarrow \Gamma', x : [\![\tau_x]\!]_\Delta \vdash [\![p_v]\!]_\Delta : [\![[x/x']\tau'_v]\!]_\Delta && \text{[subtype pres.]}
\end{aligned}$$

Since $[\![\texttt{val}\ v\ :\ \tau_v\ =\ p_v]\!]_\Delta = \{v : [\![p_v]\!]_\Delta\}$:

$$x;\Gamma', x : [\![\tau_x]\!]_\Delta \vdash \{v : [\![p_v]\!]_\Delta\} : \{v : [\![p_v]\!]_\Delta.\text{type}\}$$
$$x;\Gamma', x : [\![\tau_x]\!]_\Delta \vdash \{v : [\![p_v]\!]_\Delta\} : \{v : [\![[x/x']\tau_v']\!]_\Delta\} \qquad \text{[section 4.2.4 of [3]]}$$

– *For defs*:

$$(3) \Rightarrow \forall \texttt{def}\ f\ :\ \tau_a\ x_a \to \tau_r = e_f \in \overline{d}.$$
$$\Delta\Sigma\Gamma S \vdash n \ni_{x'} \texttt{def}\ f\ :\ \tau_a'\ x_a' \to \tau_r' \tag{4}$$
$$\Delta\Sigma\Gamma, x : \tau_x S \vdash [x/x']\tau_a' <:\ \tau_a \tag{5}$$
$$\Delta\Sigma\Gamma, x : \tau_x, x_a : \tau_a S \vdash \tau_r <:\ [x, x_a/x', x_a']\tau_r' \tag{6}$$
$$\Delta\Sigma\Gamma, x : \tau_x, x_a : \tau_a\, S \vdash e_f : \tau_{ra} \tag{7}$$
$$\Delta\Sigma\Gamma, x : \tau_x, x_a : \tau_a S \vdash \tau_{ra} <:\ \tau_r \tag{8}$$

$$(5) \Rightarrow \Gamma', x : [\![\tau_x]\!]_\Delta \vdash [\![[x/x']\tau_a']\!]_\Delta <:\ [\![\tau_a]\!]_\Delta$$
$$(6) \Rightarrow \Gamma', x : [\![\tau_x]\!]_\Delta, x_a : [\![\tau_a]\!]_\Delta \vdash [\![\tau_r]\!]_\Delta <:\ [\![[x, x_a/x', x_a']\tau_r']\!]_\Delta$$
$$(7,8) \Rightarrow \Gamma', x : [\![\tau_x]\!]_\Delta, x_a : [\![\tau_a]\!]_\Delta \vdash [\![e_f]\!]_\Delta : [\![\tau_r]\!]_\Delta$$
$$\Rightarrow \Gamma', x : [\![\tau_x]\!]_\Delta, x_a : [\![\tau_a]\!]_\Delta \vdash [\![e_f]\!]_\Delta : [\![[x, x_a/x', x_a']\tau_r']\!]_\Delta$$

Then $[\![\texttt{def}\ f\ :\ \tau_a\ x_a \to \tau_r = e_f]\!]_\Delta = \{f = \lambda(x_a : [\![\tau_a]\!]_\Delta)[\![e_f]\!]_\Delta\}$, and $\Gamma' \vdash \{f = \lambda(x_a : [\![\tau_a]\!]_\Delta)[\![e_f]\!]_\Delta\} : \{f : \forall(x_a : [\![\tau_a]\!]_\Delta)[\![\tau_r]\!]_\Delta\}$.

– *For types*:

$$(3) \Rightarrow \forall \texttt{type}\ t\ =\ \tau_t \in \overline{d}.$$
$$\Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_{x'} \texttt{type}\ t\ B'\ \tau_t'$$
$$\Delta\Sigma\Gamma, x : \tau_x S \vdash \texttt{type}\ t\ =\ \tau_t <:\ [x/x']\texttt{type}\ t\ B'\ \tau_t'$$
$$\Rightarrow \Gamma', x : [\![\tau_x]\!]_\Delta \vdash [\![\texttt{type}\ t\ =\ \tau_t]\!]_\Delta <:\ [\![[x/x']\texttt{type}\ t\ B'\ \tau_t']\!]_\Delta$$

If we gather all σ s.t. $\Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_{x'} \sigma$ and put them into $\overline{\sigma_\tau}$, then

$$\Gamma' \vdash \bigwedge_{\sigma \in \overline{\sigma_\tau}} [\![\sigma]\!]_\Delta <:\ [\![n\,\overline{r}]\!]_\Delta$$

We've already shown that

$$\forall \sigma \in \overline{\sigma_\tau}$$
$$\exists d \in \overline{d} \text{ s.t.}$$
$$x;\Gamma', x : [\![\tau_x]\!]_\Delta \vdash [\![d]\!]_\Delta : [\![\sigma]\!]_\Delta$$
and each d is distinct

By ANDDEF-I,

$$x;\Gamma', x : [\![\tau_x]\!]_\Delta \vdash \wedge_{d \in \overline{d}} [\![d]\!]_\Delta : \wedge_{\sigma \in \sigma_\tau} [\![\sigma]\!]_\Delta$$
$$\Rightarrow \Gamma' \vdash v(x : \wedge_{\sigma \in \sigma_\tau} [\![\sigma]\!]_\Delta) \wedge_{d \in \overline{d}} : \mu(x : \wedge_{\sigma \in \sigma_\tau} [\![\sigma]\!]_\Delta)$$
$$\Rightarrow \Gamma' \vdash [\![\texttt{new}\ \tau\{x \Rightarrow \overline{d}\}]\!]_\Delta : [\![\tau]\!]_\Delta$$

- *Case* T-LET:

  Let $e = \texttt{let}\ x : \tau_1 = e_1\ \texttt{in}\ e_2$.

  By the translation rules, $t' = \texttt{let}\ x = [\![e_1]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta$.

  (1) implies:

$$
\begin{aligned}
&\Delta\Sigma\Gamma S \vdash e_1 : \tau_1' && (2)\\
&\Delta\Sigma\Gamma S \vdash \tau_1' <: \tau_1 && (3)\\
&\Delta\Sigma\Gamma, x : \tau_1\, S \vdash e_2 : \tau' && (4)\\
&\Delta\Sigma\Gamma, x : \tau_1 S \vdash \tau' <: \tau && (5)\\
&x \notin fv(\tau) && (6)
\end{aligned}
$$

  By IH,

$$
\begin{aligned}
&(2) \Rightarrow \Gamma' \vdash [\![e_1]\!]_\Delta : [\![\tau_1']\!]_\Delta\\
&(3) \Rightarrow \Gamma' \vdash [\![e_1]\!]_\Delta : [\![\tau_1]\!]_\Delta\\
&(4) \Rightarrow \Gamma', x : [\![\tau_1]\!]_\Delta \vdash [\![e_2]\!]_\Delta : [\![\tau']\!]_\Delta\\
&(5) \Rightarrow \Gamma', x : [\![\tau_1]\!]_\Delta \vdash [\![e_2]\!]_\Delta : [\![\tau]\!]_\Delta\\
&(6) \Rightarrow x \notin fv([\![\tau]\!]_\Delta) \Rightarrow \Gamma' \vdash [\![e]\!]_\Delta : [\![\tau]\!]_\Delta \qquad [\textsc{Let}]
\end{aligned}
$$

$\square$

This proof depended on the lemmas: expansion preservation, access preservation, substitution preservation, and subtype preservation. They will be proved below.

Expansion preservation shows that if a pDOT term types to the translation of Nominal Wyvern type τ, then it also types to the translation of the expansion of τ.

Lemma 2 (Translation Preserves Expansion)**.** *Given well-formed top-level $\overline{D}$, if $\Delta\Sigma\Gamma S \vdash \tau \prec n\,\overline{r}$ and $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta \vdash t : [\![\tau]\!]_\Delta$, then $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta \vdash t : [\![n\,\overline{r}]\!]_\Delta$.*

This is true because expansion implies subtyping:

Lemma 3 (Expansion Implies Subtyping)**.** *Given well-formed top-level $\overline{D}$, if $\Delta\Sigma\Gamma S \vdash \beta_1\,\overline{r_1} \prec \beta_2\,\overline{r_2}$, then $\Delta\Sigma\Gamma S \vdash \beta_1\,\overline{r_1} <: \beta_2\,\overline{r_2}$.*

The rules of expansion shows that this lemma is trivially true for all cases except for TE-UPPER due to S-REFL. However, the premises for TE-UPPER are exactly the same as the premises for S-UPPER with all recursive subtype judgments replaced with recursive type expansion judgments. An induction proof on the derivation of a type induction judgment will show that subtyping holds as well. Combined with subtype preservation (proved below), we can thus prove that type expansions are preserved by the translation.

Access preservation shows that if a member σ exists in a type $n\,\overline{r}$, then the translated member exists in the translated type as well. This is clear from the translation rules since all members of a type (fields, methods, and types) are mapped one-to-one to a corresponding member in the translated type.

Lemma 4 (Translation Preserves Access)**.** *Given well-formed top-level $\overline{D}$, if $\Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \sigma$, then $\mu(x : [\![\sigma]\!]_\Delta) \in [\![n\,\overline{r}]\!]_\Delta$.*

Substitution preservation shows that substituting a variable for a path can be done before or after a translation. This is clear from the translation rules since variables and paths are unchanged.

Lemma 5 (Substitution Preserving Translation)**.** $[[\![p]\!]_\Delta/x][\![\tau]\!]_\Delta = [\![[p/x]\tau]\!]_\Delta$

Subtype preservation shows that if two types are subtypes in Nominal Wyvern, then their translations are subtypes in pDOT as well. We first prove this for the nominal subset of types.

Lemma 6 (Translation Preserves Nominal Subtyping)**.** *Given well-formed top-level $\overline{D}$, if $\Delta\Sigma\Gamma S \vdash n_1 \xrightarrow{\overline{r_1}} n_2$, then $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta \vdash [\![n_1]\!]_\Delta <:\ [\![n_2]\!]_\Delta$.*

Proof. To simplify notation, denote $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta$ with Γ'.

Assume the premise $\Delta\Sigma\Gamma S \vdash n_1 \xrightarrow{\overline{r_1}} n_2$ (1). Prove by induction on nominal subtyping:

- *Case* SN-REFL:

  This means $n_2 = n_1$, which implies $[\![n_2]\!]_\Delta = [\![n_1]\!]_\Delta$. By REFL, $\Gamma' \vdash [\![n_1]\!]_\Delta <:\ [\![n_2]\!]_\Delta$.

- *Case* SN-TRANS:

  (1) implies:

$$\begin{aligned} &\Sigma \ni n_1\, r_1' <:\ n' && (2)\\ &\Delta\Sigma\Gamma S \vdash \overline{r_1} <:\ r_1' &&\\ &\Delta\Sigma\Gamma S \vdash n' \xrightarrow{\overline{r_1}} n_2 && (3) \end{aligned}$$

  (2) implies $TL.n'$ is part of the definition of $TL.n_1$
  i.e. the definition n_1 in TL is $n_1 = ... \wedge TL.n' \wedge ....$

  By AND$_1$-<: and AND$_2$-<: , $\Gamma' \vdash [\![n_1]\!]_\Delta <:\ [\![n']\!]_\Delta$.

□

Now we can prove the general version of subtype preservation. Note that this proof depends on path typing preservation, a subset of typing preservation. There is, however, not a cyclic dependency because path typing preservation (the first 3 cases in the typing preservation proof) does not depend on subtype preservation nor the other cases of typing preservation.

Lemma 7 (Translation Preserves Subtyping)**.** *Given well-formed top-level $\overline{D}$, if $\Delta\Sigma\Gamma S \vdash \tau_1 <:\ \tau_2$, then $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta \vdash [\![\tau_1]\!]_\Delta <:\ [\![\tau_2]\!]_\Delta$.*

Proof. To simplify notation, denote $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta$ with Γ', $[\![\tau_1]\!]_\Delta$ with T_1, and $[\![\tau_2]\!]_\Delta$ with T_2.

Assume the premise $\Delta\Sigma\Gamma S \vdash \tau_1 <:\ \tau_2$ (1). Prove by induction on subtyping:

- *Case* S-TOP:

  Then $\tau_2 = \top$, so $T_2 = \top$, so $\Gamma' \vdash T_1 <: T_2$.

- *Case* S-BOT:

  Then $\tau_1 = \bot$, so $T_1 = \bot$, so $\Gamma' \vdash T_1 <: T_2$.

- *Case* S-UPPER:

  Let $\tau_1 = p.t\,\overline{r_1}$.

  (1) implies:

$$\begin{aligned}
&\Delta\Sigma\Gamma S \vdash p : \tau_p && (2)\\
&\Delta\Sigma\Gamma S \vdash \tau_p \prec n\,\overline{r} &&\\
&\Delta\Sigma\Gamma S \vdash n\,\overline{r} \ni_x \textbf{\texttt{type}}\ t \leqq \tau_t &&\\
&\Delta\Sigma\Gamma S \vdash [p/x]\tau_r\,\overline{r_1} <: \tau_2 && (3)
\end{aligned}$$

$$\begin{aligned}
(2) &\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : [\![\tau_p]\!]_\Delta && \text{[path typing pres.]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : [\![n\,\overline{r}]\!]_\Delta && \text{[expansion pres.]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : \mu(x : \{t : T ... [\![\tau_t]\!]_\Delta\}) && \text{[for some } T\text{, access pres.]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta : [[\![p]\!]_\Delta/x]\{t : T ... [\![\tau_t]\!]_\Delta\} &&\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta.t <: [[\![p]\!]_\Delta/x][\![\tau_t]\!]_\Delta && \text{[S}\textsc{el}\text{-<:]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta.t \wedge [\![\overline{r_1}]\!]_\Delta <: [[\![p]\!]_\Delta/x][\![\tau_t]\!]_\Delta && \text{[A}\textsc{nd}\text{-<: ,T}\textsc{rans}\text{-<:]}\\
&\Rightarrow \Gamma' \vdash [\![p]\!]_\Delta.t \wedge [\![\overline{r_1}]\!]_\Delta <: ([[\![p]\!]_\Delta/x][\![\tau_t]\!]_\Delta) \wedge [\![\overline{r_1}]\!]_\Delta && \text{[<: -A}\textsc{nd}\text{]}\\
&\Rightarrow \Gamma' \vdash [\![p.t\,\overline{r_1}]\!]_\Delta <: ([[\![p]\!]_\Delta/x][\![\tau_t]\!]_\Delta) \wedge [\![\overline{r_1}]\!]_\Delta &&
\end{aligned}$$

$$\begin{aligned}
\text{IH} \,\&\, (3) &\Rightarrow \Gamma' \vdash [[\![p]\!]_\Delta/x][\![\tau_t\,\overline{r_1}]\!]_\Delta <: [\![\tau_2]\!]_\Delta\\
&\Rightarrow \Gamma' \vdash ([[\![p]\!]_\Delta/x][\![\tau_t]\!]_\Delta) \wedge [\![\overline{r_1}]\!]_\Delta <: [\![\tau_2]\!]_\Delta\\
&\Rightarrow \Gamma' \vdash [\![[\![p]\!]_\Delta.t\,\overline{r_1}]\!]_\Delta <: [\![\tau_2]\!]_\Delta\\
&\text{i.e. } \Gamma' \vdash [\![\tau_1]\!]_\Delta <: [\![\tau_2]\!]_\Delta
\end{aligned}$$

- *Case* S-LOWER:

  Follows the same procedure as the S-UPPER case, except uses <: -SEL instead of SEL-<: .

- *Case* S-REFL:

  Let $\tau_1 = \beta\,\overline{r_1}$, and $\tau_2 = \beta\,\overline{r_2}$.

(1) implies:

$$
\begin{aligned}
&\Delta\Sigma\Gamma S \vdash \overline{r_1} <:\ \overline{r_2} \\
\Rightarrow&\forall \Delta\Sigma\Gamma S \vdash \overline{r_2} \ni \textbf{type}\ t\ B_2\ \tau_{t2}. \\
&\quad \exists \Delta\Sigma\Gamma S \vdash \overline{r_1} \ni \textbf{type}\ t\ B_1\ \tau_{t1}\ s.t. \\
&\quad\quad \Delta\Sigma\Gamma S \vdash \textbf{type}\ t\ B_1\ \tau_{t1} <:\ \textbf{type}\ t\ B_2\ \tau_{t2} \\
\Rightarrow&\forall \{t : S_2 ... T_2\} \in [\![\overline{r_2}]\!]_\Delta. \\
&\quad \exists \{t : S_1 ... T_1\} \in [\![\overline{r_1}]\!]_\Delta\ s.t. \\
&\quad\quad \Gamma' \vdash \{t : S_1 ... T_1\} <:\ \{t : S_2 ... T_2\} && \text{[member subtype pres.]} \\
\Rightarrow&\Gamma' \vdash [\![\overline{r_1}]\!]_\Delta <:\ [\![\overline{r_2}]\!]_\Delta \\
\Rightarrow&\Gamma' \vdash [\![\beta]\!]_\Delta \wedge [\![\overline{r_1}]\!]_\Delta <:\ [\![\beta]\!]_\Delta \wedge [\![\overline{r_2}]\!]_\Delta \\
\Rightarrow&\Gamma' [\![\beta\, \overline{r_1}]\!]_\Delta <:\ [\![\beta\, \overline{r_2}]\!]_\Delta \\
\Rightarrow&\Gamma' [\![\tau_1]\!]_\Delta <:\ [\![\tau_2]\!]_\Delta
\end{aligned}
$$

- *Case* S-NAME:

  Let $\tau_1 = n_1\, \overline{r_1}$, and $\tau_2 = n_2\, \overline{r_2}$.

  (1) implies:

$$
\begin{aligned}
&\Delta\Sigma\Gamma S \vdash n_1 \xrightarrow{\overline{r_1}} n_2 && (2) \\
&\Delta\Sigma\Gamma S \vdash \overline{r_1} <:\ \overline{r_2} && (3)
\end{aligned}
$$

  Case S-REFL implies that

$$
\begin{aligned}
(3) &\Rightarrow \Gamma' \vdash [\![\overline{r_1}]\!]_\Delta <:\ [\![\overline{r_2}]\!]_\Delta \\
&\Rightarrow \Gamma' \vdash [\![n_1]\!]_\Delta \wedge [\![\overline{r_1}]\!]_\Delta <:\ [\![\overline{r_2}]\!]_\Delta && [\text{AND}_2\text{-}<:]
\end{aligned}
$$

  Also,

$$
\begin{aligned}
(2) &\Rightarrow \Gamma' \vdash [\![n_1]\!]_\Delta <:\ [\![n_2]\!]_\Delta && \text{[nominal subtype pres.]} \\
&\Rightarrow \Gamma' \vdash [\![n_1]\!]_\Delta \wedge [\![\overline{r_1}]\!]_\Delta <:\ [\![n_2]\!]_\Delta && [\text{AND}_1\text{-}<:]
\end{aligned}
$$

  Applying <: -AND we get

$$
\Gamma' \vdash [\![n_1]\!]_\Delta \wedge [\![\overline{r_1}]\!]_\Delta <:\ [\![n_2]\!]_\Delta \wedge [\![\overline{r_2}]\!]_\Delta
$$

  i.e.

$$
\Gamma' \vdash [\![n_1\, \overline{r_1}]\!]_\Delta <:\ [\![n_2\, \overline{r_2}]\!]_\Delta
$$

$\square$

This proof depends on member subtype preservation, proved below.

Lemma 8 (Translation Preserves Member Subtype)**.** *Given well-formed top-level $\overline{D}$, if* $\Delta\Sigma\Gamma S \vdash \sigma_1 <: \sigma_2$*, then* $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta \vdash [\![\sigma_1]\!]_\Delta <: [\![\sigma_2]\!]_\Delta$.

Proof. To simplify notation, denote $\Gamma_{\overline{D}}, [\![\Gamma]\!]_\Delta, [\![S]\!]_\Delta$ with Γ', $[\![\sigma_1]\!]_\Delta$ with T_1, and $[\![\sigma_2]\!]_\Delta$ with T_2. Assume the premise $\Delta\Sigma\Gamma S \vdash \sigma_1 <: \sigma_2$ (1). Prove by induction on member subtyping:

- *Case: σ is a type member*:

  Let $\sigma_1 = \texttt{type}\ t\ B_1\ \tau_1$ and $\sigma_2 = \texttt{type}\ t\ B_2\ \tau_2$.

  - *Subcase* SS-UPPER:

    Then B_1 is $\leqq$, B_2 is $\leq$, and $\Delta\Sigma\Gamma S \vdash \tau_1 <: \tau_2$. By subtype preservation (not a cyclic dependency due to subtype decidability):

    $$\Gamma' \vdash [\![\tau_1]\!]_\Delta <: [\![\tau_2]\!]_\Delta$$

    , which implies

    $$\begin{aligned} \Rightarrow & T_1 = \{S_1 ... [\![\tau_1]\!]_\Delta\} && [S_1 \text{ is } \bot \text{ or } [\![\tau_1]\!]_\Delta] \\ & T_2 = \{\bot ... [\![\tau_2]\!]_\Delta\} \\ \Rightarrow & \Gamma' \vdash T_1 <: T_2 && [\text{TYP-<:-TYP}] \end{aligned}$$

  - *Subcase* SS-LOWER:

    Then B_1 is $\geqq$, B_2 is $\geq$, and $\Delta\Sigma\Gamma S \vdash \tau_2 <: \tau_1$. Similarly, by subtype preservation:

    $$\Gamma' \vdash [\![\tau_2]\!]_\Delta <: [\![\tau_1]\!]_\Delta$$

    , which implies

    $$\begin{aligned} \Rightarrow & T_1 = \{[\![\tau_1]\!]_\Delta ... S_1\} && [S_1 \text{ is } [\![\tau_1]\!]_\Delta \text{ or } \top] \\ & T_2 = \{[\![\tau_2]\!]_\Delta ... \top\} \\ \Rightarrow & \Gamma' \vdash T_1 <: T_2 && [\text{TYP-<:-TYP}] \end{aligned}$$

  - *Subcase* SS-EXACT:

    Then B_1 is $=$, B_2 is $=$, and $\Delta\Sigma\Gamma S \vdash \tau_1 <: \tau_2$ and $\Delta\Sigma\Gamma S \vdash \tau_2 <: \tau_1$. Similarly, by subtype preservation:

    $$\Gamma' \vdash [\![\tau_1]\!]_\Delta <: [\![\tau_2]\!]_\Delta \quad \text{and} \quad \Gamma' \vdash [\![\tau_2]\!]_\Delta <: [\![\tau_1]\!]_\Delta$$

    , which implies

    $$\begin{aligned} \Rightarrow & T_1 = \{[\![\tau_1]\!]_\Delta ... [\![\tau_1]\!]_\Delta\} \\ & T_2 = \{[\![\tau_2]\!]_\Delta ... [\![\tau_2]\!]_\Delta\} \\ \Rightarrow & \Gamma' \vdash T_1 <: T_2 && [\text{TYP-<:-TYP}] \end{aligned}$$

- *Case: σ is a field*

  Let $\sigma_1 = \textbf{val}\ v\ :\ \tau_1$ and $\sigma_2 = \textbf{val}\ v\ :\ \tau_2$.

  By the translation rules, $T_1 = \{v : [\![\tau_1]\!]_\Delta\}$ and $T_2 = \{v : [\![\tau_2]\!]_\Delta\}$.

  (1) implies:

$$
\begin{aligned}
&\Delta\Sigma\Gamma S \vdash \tau_1 <:\ \tau_2 \\
\Rightarrow &\Gamma' \vdash [\![\tau_1]\!]_\Delta <:\ [\![\tau_2]\!]_\Delta \\
\Rightarrow &\Gamma' \vdash \{v : [\![\tau_1]\!]_\Delta\} <:\ \{v : [\![\tau_2]\!]_\Delta\} && \text{[FLD-<: -FLD]}
\end{aligned}
$$

- *Case: σ is a method*

  Let $\sigma_1 = \textbf{def}\ f\ :\ \tau_{a1}\ x \to \tau_{r1}$ and $\sigma_2 = \textbf{def}\ f\ :\ \tau_{a2}\ x \to \tau_{r2}$ (method argument name pre-normalized via substitution).

  By the translation rules,

$$
\begin{aligned}
T_1 &= \{f : \forall(x : [\![\tau_{a1}]\!]_\Delta)[\![\tau_{r1}]\!]_\Delta\} \\
T_2 &= \{f : \forall(x : [\![\tau_{a2}]\!]_\Delta)[\![\tau_{r2}]\!]_\Delta\}
\end{aligned}
$$

  (1) implies:

$$
\begin{aligned}
&\Delta\Sigma\Gamma S \vdash \tau_{a2} <:\ \tau_{a1} && (2) \\
&\Delta\Sigma\Gamma, x : \tau_{a2} S \vdash \tau_{r1} <:\ \tau_{r2} && (3)
\end{aligned}
$$

$$
\begin{aligned}
(2) &\Rightarrow \Gamma' \vdash [\![\tau_{a2}]\!]_\Delta <:\ [\![\tau_{a1}]\!]_\Delta \\
(3) &\Rightarrow \Gamma', x : [\![\tau_{a2}]\!]_\Delta \vdash [\![\tau_{r1}]\!]_\Delta <:\ [\![\tau_{r2}]\!]_\Delta \\
&\Rightarrow \Gamma' \vdash \forall(x : [\![\tau_{a1}]\!]_\Delta)[\![\tau_{r1}]\!]_\Delta <:\ \forall(x : [\![\tau_{a2}]\!]_\Delta)[\![\tau_{r2}]\!]_\Delta && \text{[ALL-<: -ALL]} \\
&\Rightarrow \Gamma' \vdash T_1 <:\ T_2 && \text{[FLD-<: -FLD]}
\end{aligned}
$$

□

Since the translation is proven to preserve typing and subtyping relations, we can prove type safety by showing that every reduction step taken in Nominal Wyvern matches with a reduction step in pDOT.

The first step is to show that the Nominal Wyvern path reduction judgments ($\mu \vdash p\ \to\ l$) match pDOT's path lookup judgments. For the rest of the soundness proof, we use $\gamma \vdash s \rightsquigarrow^+ s$ to refer to a series of single step lookup judgments that transitively chain together. This gives a more flexible way of expressing $\rightsquigarrow^*$ without strictly following pDOT's explicit right-associative definition of $\rightsquigarrow^*$. As a result, we ignore the rules LOOKUP-REFL and LOOKUP-TRANS, and instead modify the remaining three rules to use $\rightsquigarrow^+$ instead of plain $\rightsquigarrow$. This makes the proof clearer while maintaining the semantics of path lookup.

Lemma 9 (Translation Preserves Path Lookup)**.** *If* $\mu \vdash p \;\rightarrow\; l$*, then* $[\![\mu]\!]_\Delta \vdash [\![p]\!]_\Delta \rightsquigarrow^+ v$ *where* $v = [\![\mu(l)]\!]_\Delta$.

Proof. Assume the premise $\mu \vdash p \;\rightarrow\; l$ (1).

Prove by induction on path lookup (1):

- *Case* EVP-LOC:

  In this case, $p = l$. Therefore $[\![p]\!]_\Delta = x_l$.

  (1) implies that $l \in \mu$. Therefore, the translation rules show that $[\![\mu]\!]_\Delta$ includes $x_l \mapsto [\![\mu(l)]\!]_\Delta$, i.e. $[\![\mu]\!]_\Delta(x_l) = [\![\mu(l)]\!]_\Delta$.

  Thus, we can apply LOOKUP-STEP-VAR to get $\gamma \vdash x_l \rightsquigarrow^+ [\![\mu(l)]\!]_\Delta$.

- *Case* EVP-PATH:

  Let $p = p'.v$. Therefore $[\![p]\!]_\Delta = p'.v$.

  (1) implies that

$$\mu \vdash p' \;\rightarrow\; l' \tag{2}$$
$$\mu(l') \ni_{x_s} \texttt{val}\; v \;:\; \tau_v \;=\; p_v \tag{3}$$
$$\mu \vdash [p'/x_s]p_v \;\rightarrow\; l \tag{4}$$

  By IH,

$$\begin{aligned}
(2) \Rightarrow & [\![\mu]\!]_\Delta \vdash [\![p']\!]_\Delta \rightsquigarrow^+ [\![\mu(l')]\!]_\Delta && (5)\\
(4) \Rightarrow & [\![\mu]\!]_\Delta \vdash [\![[p'/x_s]p_v]\!]_\Delta \rightsquigarrow^+ [\![\mu(l)]\!]_\Delta \\
\Rightarrow & [\![\mu]\!]_\Delta \vdash [[\![p']\!]_\Delta/x_s][\![p_v]\!]_\Delta \rightsquigarrow^+ [\![\mu(l)]\!]_\Delta && (6)
\end{aligned}$$

  (3) implies that $l' : \{x_s : \overline{d}\} \in \mu$ and $\texttt{val}\; v \;:\; \tau_v \;=\; p_v \in \overline{d}$. By the translation rules,

$$[\![\mu(l')]\!]_\Delta = [\![\{x_s : \overline{d}\}]\!]_\Delta$$

  , i.e.

$$[\![\mu(l')]\!]_\Delta = v(x_s : [\![sig(\overline{d})]\!]_\Delta)[\![\overline{d}]\!]_\Delta$$

  , where

$$[\![\overline{d}]\!]_\Delta \ni [\![\texttt{val}\; v \;:\; \tau_v \;=\; p_v]\!]_\Delta$$

  , i.e.

$$[\![\overline{d}]\!]_\Delta \ni \{v = [\![p_v]\!]_\Delta\}$$

  Plugging this back into (5), we get

$$[\![\mu]\!]_\Delta \vdash p' \rightsquigarrow^+ v(x_s : [\![sig(\overline{d})]\!]_\Delta)...\{v = [\![p_v]\!]_\Delta\}...$$

  Applying LOOKUP-STEP-VAL gets $[\![\mu]\!]_\Delta \vdash [\![p']\!]_\Delta.v \rightsquigarrow^+ [[\![p']\!]_\Delta/x_s][\![p_v]\!]_\Delta$. Chaining this with (6) we get $[\![\mu]\!]_\Delta \vdash [\![p']\!]_\Delta.v \rightsquigarrow^+ [\![\mu(l)]\!]_\Delta$, i.e. $[\![\mu]\!]_\Delta \vdash [\![p]\!]_\Delta \rightsquigarrow^+ [\![\mu(l)]\!]_\Delta$

$\square$

Now we can prove that each reduction step taken in Nominal Wyvern corresponds to a reduction step in pDOT.

Lemma 10 (Reduction Correspondence)**.** *If* $\Delta\Sigma \cdot S \vdash e : \tau$*,* $\Delta\Sigma \cdot S \vdash \mu$*, and* $\mu \mid e \longmapsto \mu' \mid e'$*, then* $[\![\mu]\!]_\Delta | [\![e]\!]_\Delta \longmapsto [\![\mu']\!]_\Delta | [\![e']\!]_\Delta$.

Proof. Assume the premises

$$\Delta\Sigma \cdot S \vdash e : \tau \tag{1}$$
$$\Delta\Sigma \cdot S \vdash \mu \tag{2}$$
$$\mu \mid e \longmapsto \mu' \mid e' \tag{3}$$

Prove by induction on reduction:

- *Case* EV-APP:

  Let $e = p.f(p_a)$.

  (3) implies that

  $$\mu \vdash p \rightarrow l \tag{4}$$
  $$\mu(l) \ni_{x_s} \texttt{def}\ f\ :\ \tau_x\ x \rightarrow \tau_r = e_f \tag{5}$$
  $$e' = [p, p_a/x_s, x]e_f \tag{6}$$

  By the translation rules,

  $$\begin{aligned} [\![e]\!]_\Delta &= [\![p]\!]_\Delta.f[\![p_a]\!]_\Delta \\ [\![e']\!]_\Delta &= [\![[p, p_a/x_s, x]e_f]\!]_\Delta \\ &= [[\![p]\!]_\Delta, [\![p_a]\!]_\Delta/x_s, x][\![e_f]\!]_\Delta \end{aligned}$$

  By path lookup preservation, (4) implies that $[\![\mu]\!]_\Delta \vdash [\![p]\!]_\Delta \rightsquigarrow^+ [\![\mu(l)]\!]_\Delta$ (7).

  (5) implies that $l : \{x_s : \overline{d}\} \in \mu$, and $\texttt{def}\ f\ :\ \tau_x\ x \rightarrow \tau_r = e_f \in \overline{d}$. By the translation rules,

  $$[\![\mu(l)]\!]_\Delta = [\![\{x_s : \overline{d}\}]\!]_\Delta$$

  , i.e.

  $$[\![\mu(l)]\!]_\Delta = v(x_s : [\![sig(\overline{d})]\!]_\Delta)[\![\overline{d}]\!]_\Delta$$

  , where

  $$[\![\overline{d}]\!]_\Delta \ni [\![\texttt{def}\ f\ :\ \tau_x\ x \rightarrow \tau_r = e_f]\!]_\Delta$$

  , i.e.

  $$[\![\overline{d}]\!]_\Delta \ni \{f = \lambda(x : [\![\tau_x]\!]_\Delta)[\![e_f]\!]_\Delta\}$$

  Plugging this back into (7) we get

  $$[\![\mu]\!]_\Delta \vdash [\![p]\!]_\Delta \rightsquigarrow^+ v(x_s : [\![sig(\overline{d})]\!]_\Delta)...\{f = \lambda(x : [\![\tau_x]\!]_\Delta)[\![e_f]\!]_\Delta\}...$$

Therefore, applying LOOKUP-STEP-VAL on this we get

$$[\![\mu]\!]_\Delta \vdash [\![p.f]\!]_\Delta \rightsquigarrow^+ [[\![p]\!]_\Delta / x_s]\lambda(x : [\![\tau_x]\!]_\Delta)[\![e_f]\!]_\Delta$$

Now, following the pDOT reduction step on $[\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta$ arrives at:

$$\begin{aligned}
&[\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta \\
=&[\![\mu]\!]_\Delta \mid [\![p]\!]_\Delta.f\ [\![p_a]\!]_\Delta \\
\longmapsto&[\![\mu]\!]_\Delta \mid [[\![p_a]\!]_\Delta / x][[\![p]\!]_\Delta / x_s][\![e_f]\!]_\Delta && \text{[APPLY]} \\
=&[\![\mu']\!]_\Delta \mid [\![e']\!]_\Delta
\end{aligned}$$

- *Case* EV-LET-PATH:

  Let $e = \texttt{let}\ x = p_1\ \texttt{in}\ e_2$.

  Then $e' = [p_1/x]e_2$, and $\mu = \mu'$.

  By the translation rules,

  $$\begin{aligned}
  [\![e]\!]_\Delta &= \texttt{let}\ x = [\![p_1]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta \\
  [\![e']\!]_\Delta &= [[\![p_1]\!]_\Delta / x][\![e_2]\!]_\Delta \\
  [\![\mu']\!]_\Delta &= [\![\mu]\!]_\Delta
  \end{aligned}$$

  Following the pDOT reduction step on $[\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta$ arrives at:

  $$\begin{aligned}
  &[\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta \\
  =&[\![\mu]\!]_\Delta \mid \texttt{let}\ x = [\![p_1]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta \\
  \longmapsto&[\![\mu]\!]_\Delta \mid [\![e_2]\!]_\Delta[[\![p_1]\!]_\Delta / x] && \text{[LET-PATH]} \\
  =&[\![\mu]\!]_\Delta \mid [[\![p_1]\!]_\Delta / x][\![e_2]\!]_\Delta \\
  =&[\![\mu']\!]_\Delta \mid [\![e']\!]_\Delta
  \end{aligned}$$

- *Case* EV-LET-NEW:

  Let $e = \texttt{let}\ x = \texttt{new}\ \tau\{x_s \Rightarrow \overline{d}\}\ \texttt{in}\ e_2$.

  (3) implies that

  $$l \text{ fresh in } \mu \tag{4}$$
  $$e' = [l/x]e_2 \tag{5}$$
  $$\mu' = \mu, l : \{x_s \Rightarrow \overline{d}\} \tag{6}$$

By the translation rules,

$$\begin{aligned}
[\![e]\!]_\Delta &= \texttt{let}\ x = [\![\texttt{new}\ \tau\{x_s \Rightarrow \overline{d}\}]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta \\
&= \texttt{let}\ x = [\![\{x_s \Rightarrow \overline{d}\}]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta \\
&= \texttt{let}\ x = v(x_s : [\![sig(\overline{d})]\!]_\Delta)[\![\overline{d}]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta \\
[\![e']\!]_\Delta &= [\![[l/x]e_2]\!]_\Delta \\
&= [x_l/x][\![e_2]\!]_\Delta \\
[\![\mu']\!]_\Delta &= [\![\mu]\!]_\Delta, x_l \mapsto [\![\{x \Rightarrow \overline{d}\}]\!]_\Delta \\
&= [\![\mu]\!]_\Delta, x_l \mapsto v(x_s : [\![sig(\overline{d})]\!]_\Delta)[\![\overline{d}]\!]_\Delta
\end{aligned}$$

Following the pDOT reduction step on $[\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta$ arrives at:

$$\begin{aligned}
& [\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta \\
= & [\![\mu]\!]_\Delta \mid \texttt{let}\ x = v(x_s : [\![sig(\overline{d})]\!]_\Delta)[\![\overline{d}]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta \\
\longmapsto & [\![\mu]\!]_\Delta, x \mapsto v(x_s : [\![sig(\overline{d})]\!]_\Delta)[\![\overline{d}]\!]_\Delta \mid [\![e_2]\!]_\Delta && [\text{LET-VALUE}, x \notin [\![\mu]\!]_\Delta] \\
= & [x/x_l][\![\mu']\!]_\Delta \mid [\![e_2]\!]_\Delta \\
= & [\![\mu']\!]_\Delta \mid [x_l/x][\![e_2]\!]_\Delta \\
= & [\![\mu']\!]_\Delta \mid [\![e']\!]_\Delta
\end{aligned}$$

The reduction step is legal because x is guaranteed to not be a key in $[\![\mu]\!]_\Delta$ because μ only contains ls, so $[\![\mu]\!]_\Delta$ only contains x_ls.

- *Case* EV-LET:

  Let $e = \texttt{let}\ x = e_1\ \texttt{in}\ e_2$.

  (3) implies

  $$e' = \texttt{let}\ x = e_1'\ \texttt{in}\ e_2 \tag{4}$$

  $$e_1 \text{ not a path or new expression} \tag{5}$$

  $$\mu \mid e_1 \ \longmapsto\ \mu' \mid e_1' \tag{6}$$

  By the translation rules,

  $$\begin{aligned}
  [\![e]\!]_\Delta &= \texttt{let}\ x = [\![e_1]\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta \\
  [\![e']\!]_\Delta &= \texttt{let}\ x = [\![e_1']\!]_\Delta\ \texttt{in}\ [\![e_2]\!]_\Delta
  \end{aligned}$$

  By IH,

  $$(6) \Rightarrow [\![\mu]\!]_\Delta \mid [\![e_1]\!]_\Delta \longmapsto [\![\mu']\!]_\Delta \mid [\![e_1']\!]_\Delta \tag{7}$$

Following the pDOT reduction step on $[\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta$ arrives at:

$$
\begin{aligned}
& [\![\mu]\!]_\Delta \mid [\![e]\!]_\Delta \\
=& [\![\mu]\!]_\Delta \mid \mathbf{let}\ x = [\![e_1]\!]_\Delta\ \mathbf{in}\ [\![e_2]\!]_\Delta \\
\longmapsto& [\![\mu]\!]_\Delta \mid \mathbf{let}\ x = [\![e_1']\!]_\Delta\ \mathbf{in}\ [\![e_2]\!]_\Delta && [\textsc{Ctx}, (7)] \\
=& [\![\mu']\!]_\Delta \mid [\![e']\!]_\Delta
\end{aligned}
$$

□

Reduction correspondence together with type preservation of translation shows that every well-typed program in Nominal Wyvern is well-typed in pDOT, and that every time the translated program takes a step, the original program takes the same step (i.e. reaching the same result after every step). Therefore, since pDOT's type safety guarantees that reduction will either diverge or stop at some value, we can also guarantee that Nominal Wyvern reduction of the original program will either diverge or stop at some irreducible value at the same time. In other words, evaluation will not go wrong.

Theorem 1 (Nominal Wyvern is Type Safe)**.** *For any well-typed Nominal Wyvern program, term reduction does not get stuck.*

4 Conclusion

This technical report provides an updated version of Nominal Wyvern with dynamic semantics, and proves type safety of Nominal Wyvern by providing a type preserving translation from Nominal Wyvern to pDOT, a DOT-based language already proven to be sound.

References

[1] Ligia Nistor, Darya Kurilova, Stephanie Balzer, Benjamin Chung, Alex Potanin, and Jonathan Aldrich. Wyvern: A simple, typed, and pure object-oriented language. In *Proceedings of the 5th Workshop on MechAnisms for SPEcialization, Generalization and inHerItance*, MASPEGHI '13, pages 9–16, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2046-7. doi: 10.1145/2489828.2489830. URL `http://doi.acm.org/10.1145/2489828.2489830`. 1

[2] Martin Odersky, Vincent Cremet, Christine Röckl, and Matthias Zenger. A nominal theory of objects with dependent types. In *ECOOP*, 2003. 1

[3] Marianna Rapoport and Ondřej Lhoták. A path to dot: formalizing fully path-dependent types. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):145:1–145:29, Oct 2019. doi: 10.1145/3360571. 3, 3

[4] Yu Xiang Zhu. Nominal wyvern: Employing semantic separation for usability. Master's thesis, Carnegie Mellon University, 4 2019. 1, 2