1. Design Decisions for Nuprl’s Type Theory
2. Product, Union, and List Types
3. The Curry-Howard Isomorphism, formally
4. Empty and Unit Types
Design Decisions for Nuprl’s Type Theory

• Syntax:
  – Expressions will be represented in a uniform term syntax
  – Term display is independent of the internal syntax

• Semantics:
  – Semantics models proof, not denotation
  – Semantics is based on judgments and lazy evaluation of noncanonical terms
  – Judgments concern typehood, type equality, membership, and typed equality

• Proof Theory:
  – Proofs proceed by applying sequent-style refinement rules
  – A judgment “t is a member of T” is represented as $T_{\text{ext } t}$
  – Propositions are represented as types
    Basic propositions have $\text{Ax}$ as only member
  – Typehood is represented by a cumulative hierarchy of universes

See Appendix A of the Nuprl 5 manual for details
Syntax:
  Canonical: \( S \times T, \langle e_1, e_2 \rangle \)
  Noncanonical: let \( \langle x, y \rangle = e \) in \( u \)

Evaluation:
\[
e \downarrow \langle e_1, e_2 \rangle \quad u[e_1, e_2 / x, y] \downarrow val \\
\text{let } \langle x, y \rangle = e \text{ in } u \downarrow val
\]

Semantics:
  \( S \times T \) is a type if \( S \) and \( T \) are
  \( \langle e_1, e_2 \rangle = \langle e_1', e_2' \rangle \) in \( S \times T \) if \( S \times T \) type, \( e_1 = e_1' \) in \( S \), and \( e_2 = e_2' \) in \( T \)

Library Concepts: \( e.1, e.2 \)

See Appendix A.3.2 and the library theory \texttt{core.2} for further details
Lists: Basic Data Containers

Syntax:
- Canonical: \( T \text{ list}, \; [], \; e_1::e_2 \)
- Noncanonical: \( \text{list}\_\text{ind}(e; \; \text{base}; \; x, l, f_{xl}. \text{up}) \)

Evaluation:
- \( \frac{e \downarrow []}{\frac{\text{list}\_\text{ind}(e; \; \text{base}; \; x, l, f_{xl}. \text{up}) \downarrow \text{val}}{\text{base} \downarrow \text{val}}} \)
- \( \frac{e \downarrow e_1::e_2}{\frac{\text{up}[e_1, e_2] \text{list}\_\text{ind}(e_2; \; \text{base}; \; x, l, f_{xl}. \text{up}) / x, , l, f_{xl} \downarrow \text{val}}{\frac{\text{list}\_\text{ind}(e; \; \text{base}; \; x, l, f_{xl}. \text{up}) \downarrow \text{val}}{\text{base} \downarrow \text{val}}} \)

Semantics:
- \( T \text{ list} \) is a type if \( T \) is
- \( [] = [] \) in \( T \text{ list} \) if \( T \text{ list} \) is a type
- \( e_1::e_2 = e_1'::e_2' \) in \( T \text{ list} \) if \( T \text{ list} \) type, \( e_1=e_1' \) in \( T \), and \( e_2=e_2' \) in \( T \text{ list} \)

Library Concepts:
- \( \text{hd}(e), \; \text{tl}(e), \; e_1@e_2, \; \text{length}(e), \; \text{map}(f; e), \; \text{rev}(e), \; e[i], \; e[i..j^-], \ldots \)

See Appendix A.3.10 and the library theory \texttt{list\_1} for further details
Disjoint Union: Case Distinctions

Syntax:
Canonical: \( S+T , \text{ inl}(e) , \text{ inr}(e) \)
Noncanonical: case \( e \) of inl\((x) \) \( \mapsto u \) | inr\((y) \) \( \mapsto v \)

Evaluation:
\[
\begin{align*}
\text{case } e \text{ of inl}(x) & \mapsto u \mid\text{ inr}(y) \mapsto v \downarrow \text{val} \\
\quad \quad e \downarrow \text{inl}(e') \quad u[e'/x] & \downarrow \text{val} \\
\quad \quad e \downarrow \text{inr}(e') \quad v[e'/y] & \downarrow \text{val}
\end{align*}
\]

Semantics:
\begin{itemize}
\item \( S+T \) is a type if \( S \) and \( T \) are
\item \( \text{inl}(e) = \text{inl}(e') \) in \( S+T \) if \( S+T \) type, \( e = e' \) in \( S \)
\item \( \text{inr}(e) = \text{inr}(e') \) in \( S+T \) if \( S+T \) type, \( e = e' \) in \( T \)
\end{itemize}

Library Concepts: ——

See Appendix A.3.3 for further details
### The Curry-Howard Isomorphism, Formally

<table>
<thead>
<tr>
<th>Proposition</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \land Q$</td>
<td>$P \times Q$</td>
</tr>
<tr>
<td>$P \lor Q$</td>
<td>$P + Q$</td>
</tr>
<tr>
<td>$P \Rightarrow Q$</td>
<td>$P \rightarrow Q$</td>
</tr>
<tr>
<td>$\lnot P$</td>
<td>$P \rightarrow \text{void}$</td>
</tr>
<tr>
<td>$\exists x:T. P[x]$</td>
<td>$x:T \times P[x]$</td>
</tr>
<tr>
<td>$\forall x:T. P[x]$</td>
<td>$x:T \rightarrow P[x]$</td>
</tr>
</tbody>
</table>

Need an **empty type** to represent “falsehood”

Need **dependent types** to represent quantifiers

See the library theory **core_1** for further details
**Empty Type** void

**Syntax:**
- Canonical: `void` — *no canonical elements*
- Noncanonical: `any(e)`

**Evaluation:** — *no reduction rules* —

**Semantics:**
- `void` is a type
- `e = e’` in void *never holds*

**Library Concepts:** —

See Appendix A.3.6 and Section 3 of the 1993 CS611 notes for further details

**Warning:** rules for `void` allows proving semantical nonsense like

\[ x: \text{void} \vdash 0=1 \in 2 \quad \text{or} \quad \vdash \text{void} \rightarrow 2 \text{ type} \]
Unit: ONE ELEMENT TYPE

Syntax:
  Canonical: \texttt{Unit, Ax}
  Noncanonical: \texttt{no noncanonical expressions}

Evaluation: \texttt{no reduction rules}

Semantics:
  \begin{itemize}
    \item \texttt{Unit is a type}
    \item \texttt{Ax = Ax in Unit}
  \end{itemize}

Library Concepts: ——

\texttt{Defined type in NUPRL, see the library theory core_1 for further details}