Formalising Configuration Languages

Why is this important in practice?

A precise definition enables us to create multiple implementations of compiler which are truly compatible and “correct”.

A precise definition can be used to implement supporting tools such as IDEs, graphical tools, analysers, etc.

Formalisation process gives us a deeper understanding of the language and highlights problems with the language design.

Based on the formal semantics, we can prove properties of the configuration making it highly reliable.

Formal semantics allows others people to experiment with compatible language extensions.

Does “herry” have access to machine C?

\[
\begin{align*}
A & \text{ extends } \\
& \{ \\
& \text{ allow "x";} \\
& \text{ deny "y";} \\
& \} \\
B & \text{ extends } A \\
& \{ \\
& \text{ deny "paul";} \\
& \} \\
C & \text{ extends } B \\
& \{ \\
& \text{ deny "herry";} \\
& \text{ allow "all";} \\
& \} \\
\end{align*}
\]

\[
\text{put} : S \times I \times V \rightarrow S
\]

\[
\text{put}(\emptyset_S, id, v) := \langle id, v \rangle :: \emptyset_S
\]

\[
\text{put}(\langle id, v_S \rangle :: s', id, v) := \langle id, v \rangle :: s'
\]

\[
\text{put}(\langle id_s, v_s \rangle :: s', id, v) := \langle id_s, v_s \rangle :: \text{put}(s', id, v)
\]