LaTeX-Hacking with Isabelle

Christian Urban

Document Preparation: Rough Picture

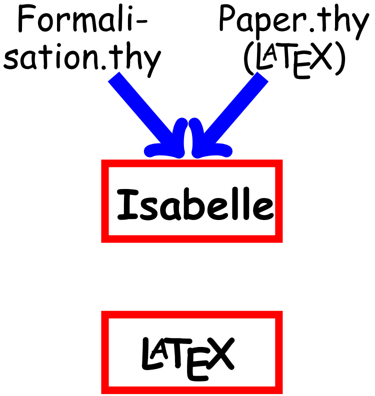
Formalisation.thy

Paper.thy (LaTeX)

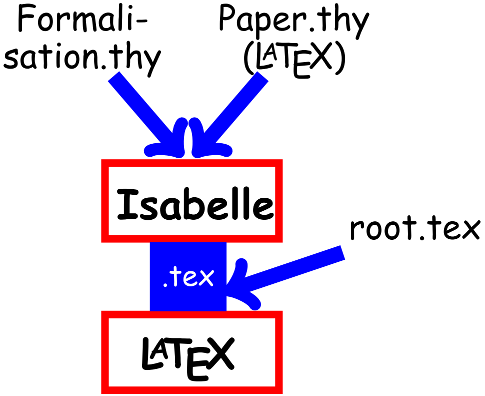
Isabelle

LaTeX

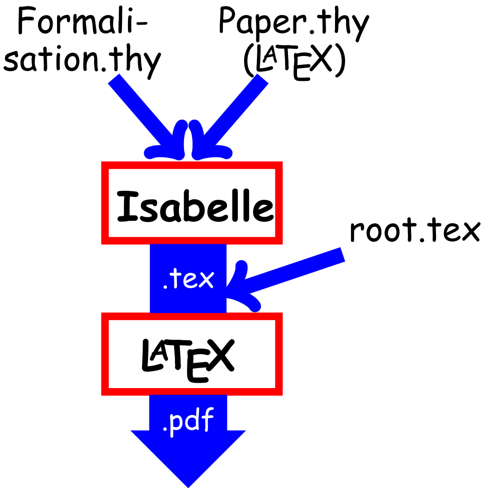
Document Preparation: Rough Picture



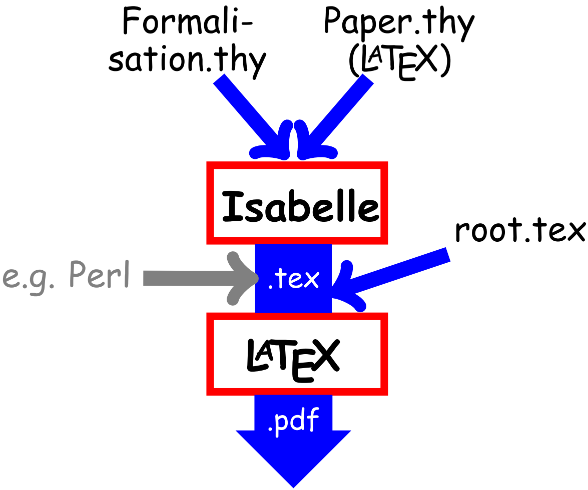
Document Preparation: Rough Picture



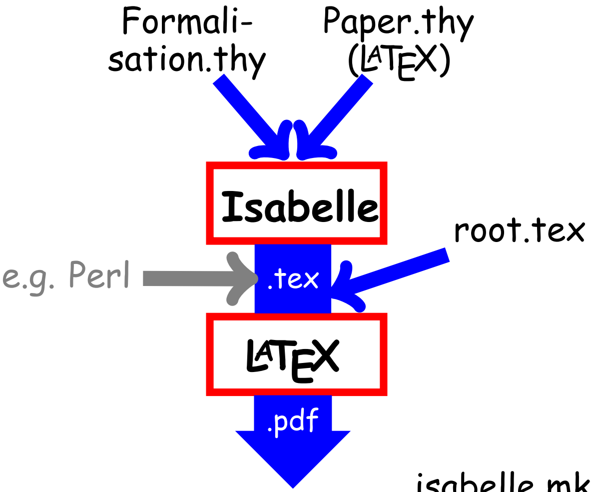
Document Preparation: Rough Picture



Document Preparation: Rough Picture



Document Preparation: Rough Picture



isabelle mkdir Paper

Sources

```
text {* ... *}
text_raw {* ... *}

fun
  rev
where
  "rev [] = []"     --"..."
| "rev (x#xs) = rev xs @ [x]"

lemma foo:
assumes a: "∀ i ∈ set Γ1. i ∈ set Γ2"
shows "set Γ1 ⊆ set Γ2"
using a
txt {* ... *}
txt_raw {* ... *}
by auto
```

Document Antiquotations

```
lemma foo:
  fixes Γ₁ Γ₂::"nat list"
  assumes a: "∀ i ∈ set Γ₁. i ∈ set Γ₂"
  shows "set Γ₁ ⊆ set Γ₂"
  using a by auto
```

Document Antiquotations

lemma foo:
  fixes Γ_1 Γ_2::"nat list"
  assumes a: "$\forall$ i $\in$ set Γ_1. i $\in$ set Γ_2"
  shows "set $\Gamma_1 \subseteq$ set Γ_2"
  using a **by** auto

You can refer inside text {*...*} to this lemma using the document antiquotation @{thm foo}.

$$\forall\, i \in \text{set } ?\Gamma_1.\ i \in \text{set } ?\Gamma_2 \Longrightarrow \text{set } ?\Gamma_1 \subseteq \text{set } ?\Gamma_2$$

Document Antiquotations

lemma foo:
  fixes Γ_1 Γ_2::"nat list"
  assumes a: "$\forall$ i $\in$ set Γ_1. i $\in$ set Γ_2"
  shows "set Γ_1 $\subseteq$ set Γ_2"
  using a **by** auto

You can refer inside text {*...*} to this lemma using the document antiquotation @{thm foo[no_vars]}.

$$\forall\, i\in \text{set}\ \Gamma_1.\ i \in \text{set}\ \Gamma_2 \Longrightarrow \text{set}\ \Gamma_1 \subseteq \text{set}\ \Gamma_2$$

Document Antiquotations

```
lemma foo:
  fixes Γ₁ Γ₂::"nat list"
  assumes a: "∀ i ∈ set Γ₁. i ∈ set Γ₂"
  shows "set Γ₁ ⊆ set Γ₂"
  using a by auto
```

You can refer inside text {*...*} to this lemma using the document antiquotation @{thm foo[no_vars]}.

$$\forall i{\in}\mathrm{set}\ \Gamma_1.\ i \in \mathrm{set}\ \Gamma_2 \Longrightarrow \mathrm{set}\ \Gamma_1 \subseteq \mathrm{set}\ \Gamma_2$$

```
notation (output) set ("_")
```

$$\forall i{\in}\Gamma_1.\ i \in \Gamma_2 \Longrightarrow \Gamma_1 \subseteq \Gamma_2$$

Changing the Order of Arguments

```
lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp
```

Changing the Order of Arguments

```
lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp

abbreviation
  my_append
where
  "my_append xs ys ≡ ys @ xs"

notation (output) my_append ("_ @ _")
```

Believe it or not, this [y] @ [x] = [x, y] is proved by Isabelle.

LaTeXsugar and Modes

```
inductive
  even and odd
where
  r1: "even 0"
| r2: "odd n ⟹ even (Suc n)"
| r3: "even n ⟹ odd (Suc n)"
```

You can print them nicely by using modes defined in LaTeXsugar.thy.

@{thm[mode=Axiom] r1[no_vars]}	@{thm[mode=Rule] r2[no_vars]}	@{thm[mode=Rule] r3[no_vars]}
$\dfrac{}{\text{even } 0}$	$\dfrac{\text{odd } n}{\text{even } (\text{Suc } n)}$	$\dfrac{\text{even } n}{\text{odd } (\text{Suc } n)}$

LaTeXsugar and Modes

```
inductive
  even and odd
where
  r1: "even 0"
| r2: "odd n ⟹ even (Suc n)"
| r3: "even n ⟹ odd (Suc n)"
```

You can print them nicely by using modes defined in LaTeXsugar.thy.

@{thm[mode=Axiom] r1[no_vars]}	@{thm[mode=Rule] r2[no_vars]}	@{thm[mode=Rule] r3[no_vars]}
$\dfrac{}{\text{even } 0}$	$\dfrac{\text{odd } n}{\text{even } (\text{Suc } n)}$	$\dfrac{\text{even } n}{\text{odd } (\text{Suc } n)}$

@{thm[mode=IfThen] r2[no_vars]}:

If odd n then even (Suc n).

Other Document Antiquotations

```
lemma disj_swap:
  shows "P ∨ Q ⟹ Q ∨ P"
apply(erule disjE)
```

1. $P \Longrightarrow Q \vee P$
2. $Q \Longrightarrow Q \vee P$

Other Document Antiquotations

```
lemma disj_swap:
  shows "P ∨ Q ⟹ Q ∨ P"
apply(erule disjE)
```

1. $P \Longrightarrow Q \vee P$
2. $Q \Longrightarrow Q \vee P$

```
lemma disj_swap:
   shows "P ∨ Q ⟹ Q ∨ P"
apply(erule disjE)
txt_raw {* @{subgoals [display]} *}
(*<*)oops(*>*)
```

Your Own Document Antiquotations

```
fun check_file_exists _ name =
  (if File.exists (Path.append
                    (Path.explode ("~~/src")) (Path.explode name))
   then ThyOutput.output [Pretty.str name]
   else
   error ("Source file " ^ (quote name) ^ " does not exist."))

val _ = ThyOutput.antiquotation "ML_file"
            (Scan.lift Args.name) check_file_exists
```

Writing @{ML_file "HOL/HOL.thy"}, produces:

HOL/HOL.thy

Your Own Theorem-Styles

lemma foo: **shows** "$\forall$ x y z. P x y z" **sorry**

```
fun strip_alls ctxt trm =
  case trm of
     Const("Trueprop", _) $ t => strip_alls ctxt t
   | Const("All", _) $ Abs(n, T, t) =>
          strip_alls ctxt (subst_bound (Free (n, T), t))
   | _ => trm
```

setup {* TermStyle.add_style "no_alls" strip_alls *}

Now @{thm_style no_alls foo} produces:

P x y z

Correct Tabulation

```
inductive
  even and odd
where
  r1: "even 0"
| r2: "odd n ⟹ even (Suc n)"
| r3: "even n ⟹ odd (Suc n)"
```

Correct Tabulation

```
inductive
  even and odd
where
  r1: "even 0"
| r2: "odd n ⟹ even (Suc n)"
| r3: "even n ⟹ odd (Suc n)"
```

```
inductive
  even and odd
where
  r1:   "even 0"
| r2:   "odd n ⟹ even (Suc n)"
| r3:   "even n ⟹ odd (Suc n)"
```

```
inductive
  even and odd
where
  r1: --"some comment"  "even 0"
| r2: --"some other comment"  "odd n ⟹ even (Suc n)"
| r3: --"something entirely else"  "even n ⟹ odd (Suc n)"
```

```
inductive
  even and odd
where
  r1: --"some comment"  "even 0"
| r2: --"some other comment"  "odd n ⟹ even (Suc n)"
| r3: --"something entirely else"  "even n ⟹ odd (Suc n)"
```

```
\renewcommand{\isamarkupcmt}[1]%
{\ifthenelse{\equal{TABSET}{#1}}{\=}%
  {\ifthenelse{\equal{TAB}{#1}}{\>}
    {\isastylecmt--- #1}}%
}%

\newenvironment{isatabbing}%
{\renewcommand{\isanewline}{\\}\begin{tabbing}}%
{\end{tabbing}}
```

Correct Tabulation

```
text_raw {*\begin{isatabbing}*}
inductive
  even and odd
where
  r1: --TABSET  "even 0"
| r2: --TAB  "odd n ⟹ even (Suc n)"
| r3: --TAB  "even n ⟹ odd (Suc n)"
text_raw {*\end{isatabbing}*}
```

```
inductive
  even and odd
where
  r1:   "even 0"
| r2:   "odd n ⟹ even (Suc n)"
| r3:   "even n ⟹ odd (Suc n)"
```

Definitions Twice?

```
inductive
  even and odd
where
  r1:  "even 0"
| r2:  "odd n ⟹ even (Suc n)"
| r3:  "even n ⟹ odd (Suc n)"
```

```
inductive
  even and odd
where
  r1:  "even 0"
| r2:  "odd n ⟹ even (Suc n)"
| r3:  "even n ⟹ odd (Suc n)"
```

Definitions Twice?

```
inductive
  even and odd
where
  r1:  "even 0"
| r2:  "odd n ⟹ even (Suc n)"
| r3:  "even n ⟹ odd (Suc n)"

inductive
  evenι and oddι
where
  r1ι:  "evenι 0"
| r2ι:  "oddι n ⟹ evenι (Suc n)"
| r3ι:  "evenι n ⟹ oddι (Suc n)"
```

Redefine in root.tex: \renewcommand{\isasymiota}{}

Slides with Beamer

```
text_raw {*
\begin{frame}
\frametitle{FooBar Slide}

 *}




text_raw {*

\end{frame}
*}
```

Slides with Beamer

```
text_raw {*
\begin{frame}
\frametitle{FooBar Slide}
\onslide<2,4>{
 *}
  lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp
text_raw {*
}
\end{frame}
*}
```

FooBar Slide

FooBar Slide

```
lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp
```

FooBar Slide

FooBar Slide

```
lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp
```

FooBar Slide

```
lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp
```

```
\definecolor{isacol:blue}{rgb}{0,0,.803}
\newcommand{\bluecmd}[1]
     {\color{isacol:blue}{\bfseries{#1}}}
\renewcommand{\isakeyword}[1]{\bluecmd{#1}}
```

FooBar Slide

```
lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp
```

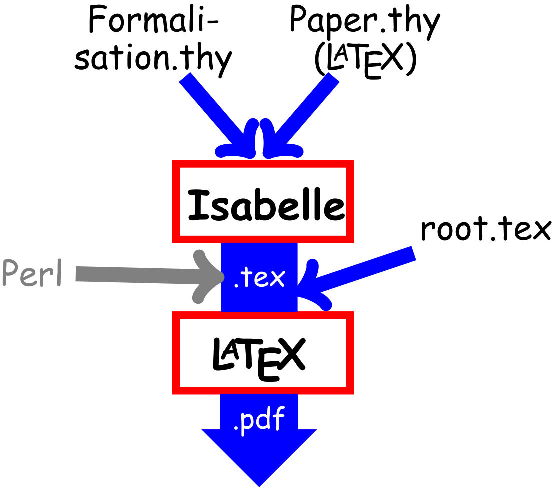
```
\renewcommand{\isakeyword}[1]{%
\ifthenelse{\equal{#1}{show}}{\browncmd{#1}}{%
\ifthenelse{\equal{#1}{case}}{\browncmd{#1}}{%
\ifthenelse{\equal{#1}{assume}}{\browncmd{#1}}{%
\ifthenelse{\equal{#1}{obtain}}{\browncmd{#1}}{%
\ifthenelse{\equal{#1}{fix}}{\browncmd{#1}}{%
\ifthenelse{\equal{#1}{oops}}{\redcmd{#1}}{%
\ifthenelse{\equal{#1}{thm}}{\redcmd{#1}}{%
{\bluecmd{#1}}}}}}}}}}%
```

FooBar Slide

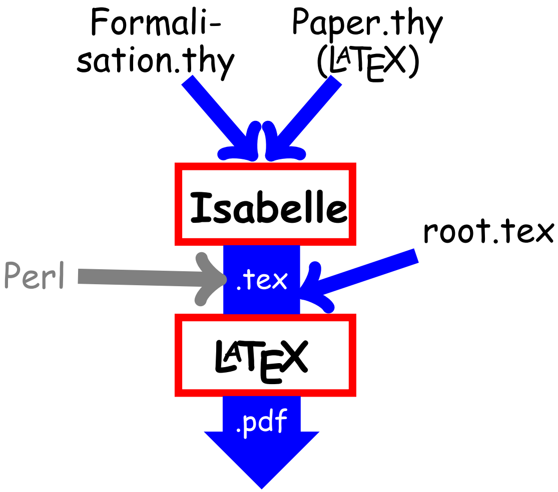
```
lemma append_bar:
  fixes x y::"nat"
  shows "[x] @ [y] = [x,y]" by simp
```

```
\definecolor{isacol:blue}{rgb}{0,0,.803}
\newcommand{\bluecmd}[1]
     {\color{isacol:blue}{\bfseries{#1}}}
\renewcommand{\isakeyword}[1]{\bluecmd{#1}}
```

A Perl Script



A Perl Script



\isachardoublequoteopen ... \isachardoublequoteclose

A Perl Script (2)

My root.tex defines the environment:

```
\newenvironment{innerdouble}%
{\isachardoublequoteopen\color{isacol:green}}%
{\color{isacol:black}\isachardoublequoteclose}
```

and the IsaMakefile calls

```
perl -i -p -e "s/..isachardoublequoteopen./\\\begin{innerdouble}/g"
Slides/generated/Slides.tex
```

and the same for isachardoublequoteclose.

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  have "Suc (Suc n) = 2 * (Suc k)" using eq by simp
  then have "∃ k. Suc (Suc n) = 2 * k" by blast
  then show "2 DVD (Suc (Suc n))" by (simp add: divide_def)
qed
```

```
lemma even_divide:
  assumes a: "even n"
  shows "2 DVD n"
using a
proof (induct)
  case eZ
  have "0 = 2 * (0::nat)" by simp
  then show "2 DVD 0" by (auto simp add: divide_def)
next
  case (eSS n)
  have "2 DVD n" by fact
  then have "∃ k. n = 2 * k" by (simp add: divide_def)
  then obtain k where eq: "n = 2 * k" by (auto)
  h
  t
  t
qe
```

```
txt_raw {* \begin{colormixin}{20!averagebackgroundcolor} *}
...
txt_raw {* \end{colormixin} *}
```

Believe It or Not: Animations with LaTeX

$\forall \Gamma\, x\, \tau\,.\ (x\!:\!\tau) \in \Gamma \wedge \mathsf{valid}\,\Gamma \Rightarrow P\ \Gamma\,(x)\,\tau$

$\forall \Gamma\, t_1\, t_2\, \sigma\, \tau\,.$
$P\,\Gamma\, t_1\,(\sigma \rightarrow \tau) \wedge P\,\Gamma\, t_2\, \sigma$
$\Rightarrow P\,\Gamma\,(t_1\, t_2)\,\tau$

$\forall \Gamma\, x\, t\, \sigma\, \tau\,.$
$x \,\#\, \Gamma \ \wedge$
$P\,((x\!:\!\sigma)\!::\!\Gamma)\, t\, \tau \Rightarrow P\,\Gamma(\lambda x.t)\,(\sigma \rightarrow \tau)$

$$\Gamma \vdash t : \tau \Rightarrow P\,\Gamma\, t\, \tau$$

Believe It or Not: Animations with LaTeX

$$\forall \Gamma\, x\, \tau\,.\ (x\!:\!\tau) \in \Gamma \wedge \mathsf{valid}\,\Gamma \Rightarrow P\ \Gamma\,(x)\,\tau$$

$$\forall \Gamma\, t_1\, t_2\, \sigma\, \tau\,.$$
$$P\,\Gamma\,t_1\,(\sigma \rightarrow \tau) \wedge P\,\Gamma\,t_2\,\sigma$$
$$\Rightarrow P\,\Gamma\,(t_1\,t_2)\,\tau$$

$$\forall \Gamma\, x\, t\, \sigma\, \tau\,.$$
$$x \# \Gamma \ \wedge$$
$$P\,((x\!:\!\sigma)\!::\!\Gamma)\,t\,\tau \Rightarrow P\,\Gamma(\lambda x.t)\,(\sigma \rightarrow \tau)$$

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$
$$\Gamma \vdash t : \tau \Rightarrow P\,\Gamma\,t\,\tau$$

The relevant LaTeX-package is animate.sty.