CS 671 Automated Reasoning

Dependent Types

The Curry-Howard Isomorphism, again

Proposition		Type
$P \wedge Q$	$\equiv$	$P \times Q$
$P \vee Q$	$\equiv$	$P + Q$
$P \Rightarrow Q$	$\equiv$	$P \rightarrow Q$
$\neg P$	$\equiv$	$P \rightarrow \mathsf{void}$
$\exists x{:}T\,.\,P[x]$	$\equiv$	$x{:}T \times P[x]$
$\forall x{:}T\,.\,P[x]$	$\equiv$	$x{:}T \rightarrow P[x]$

Need dependent types to represent quantifiers

Why dependent types ?

- **Represent logical quantifiers as type constructs**
- **Type functions like** `λx. if x=0 then λx.x else λx,y.x`
- **Express mathematical concepts such as finite automata**
  - (Q,Σ,q_0,δ,F), where $q_0 \in Q$, $\delta{:}Q\times\Sigma\rightarrow Q$, $F\subseteq Q$.
- **Represent dependent structures in programming languages**
  - Record types $[f_1{:}T_1;\ \ldots;\ f_n{:}T_n]$
  - Variant records

    `type date = January of 1..31 | February of 1..28 | ...`
- **Nuprl had them from the beginning**
  - Other systems have recently adopted them (PVS, SPECWARE, ...)

Subsumes (independent) cartesian product

$\exists$ generalizes $\wedge$

Syntax:

Canonical: $x{:}S{\times}T$, $\langle e_1, e_2\rangle$

Noncanonical: $\mathsf{let}\ \langle x, y\rangle = e\ \mathsf{in}\ u$

Evaluation:

$$\frac{e \downarrow \langle e_1, e_2\rangle \qquad u[e_1, e_2 / x, y] \downarrow val}{\mathsf{let}\ \langle x, y\rangle = e\ \mathsf{in}\ u \downarrow val}$$

Semantics:

· $x{:}S{\times}T$ is a type if S is a type and $T[e/x]$ is a type for all e in S

· $\langle e_1, e_2\rangle = \langle e_1', e_2'\rangle$ in $x{:}S{\times}T$ if $x{:}S{\times}T$ type,

$e_1{=}e_1'$ in S, and $e_2{=}e_2'$ in $T[e_1/x]$

Dependent Products: Changes in inference rules

$\Gamma \vdash x{:}S \times T_1$ type $\lfloor$ext Ax$\rfloor$
by $\times$-R
$\Gamma \vdash S$ type $\lfloor$ext Ax$\rfloor$
$\Gamma, x'{:}S \vdash T[x'/x]$ type $\lfloor$ext Ax$\rfloor$

$\Gamma \vdash S \times T_1$ type $\lfloor$ext Ax$\rfloor$
by $\times$-R
$\Gamma \vdash S$ type $\lfloor$ext Ax$\rfloor$
$\Gamma \vdash T$ type $\lfloor$ext Ax$\rfloor$

$\Gamma \vdash x{:}S \times T$ $\lfloor$ext $\langle s, t \rangle\rfloor$
by pair-formation s
$\Gamma \vdash s \in S$ $\lfloor$ext Ax$\rfloor$
$\Gamma \vdash T[s/x]$ $\lfloor$ext $t\rfloor$
$\Gamma, x'{:}S \vdash T[x'/x]$ type $\lfloor$ext Ax$\rfloor$

$\Gamma \vdash S \times T$ $\lfloor$ext $\langle s, t \rangle\rfloor$
by pair-formation
$\Gamma \vdash S$ $\lfloor$ext $s\rfloor$
$\Gamma \vdash T$ $\lfloor$ext $t\rfloor$

$\Gamma \vdash \langle s_1, t_1 \rangle = \langle s_2, t_2 \rangle \in x{:}S \times T$ $\lfloor$ext Ax$\rfloor$
by pair-Eq
$\Gamma \vdash s_1 = s_2 \in S$ $\lfloor$ext Ax$\rfloor$
$\Gamma \vdash t_1 = t_2 \in T[s_1/x]$ $\lfloor$ext Ax$\rfloor$
$\Gamma, x'{:}S \vdash T[x'/x]$ type $\lfloor$ext Ax$\rfloor$

$\Gamma \vdash \langle s_1, t_1 \rangle = \langle s_2, t_2 \rangle \in S \times T$ $\lfloor$ext Ax$\rfloor$
by pair-Eq
$\Gamma \vdash s_1 = s_2 \in S$ $\lfloor$ext Ax$\rfloor$
$\Gamma \vdash t_1 = t_2 \in T$ $\lfloor$ext Ax$\rfloor$

WELL-FORMEDNESS

- **Rules for dependent type require checking**
  $x' : S \vdash T[x'/x]$ `type`
  - T is a function from S to types
  - T could involve complex computations,
    e.g. $T[i] \equiv$ `if` $\texttt{M}_i\texttt{(i)}\downarrow$ `then` $\mathbb{N}$ `else Void`

$\Rightarrow$ **Well-formedness is undecidable in theories with dependent types**

- **Programming languages must restrict dependencies**
  - Only allow finite dependencies $\leadsto$ decidable typechecking
- **Typechecking in NUPRL cannot be fully automated**
  - Typechecking becomes part of the proof process $\leadsto$ heuristic typechecking
- **Additional problem**
  - What is the type of a function from $\mathbb{N}$ to types? $\leadsto$ Girard Paradox

Dependent Products: Further Inference rules

$\Gamma \vdash \mathsf{let}\ \langle x_1, y_1\rangle = e_1\ \mathsf{in}\ t_1 = \mathsf{let}\ \langle x_2, y_2\rangle = e_2\ \mathsf{in}\ t_2 \in C[e_1/z]$ ⌊ext Ax⌋

by `spreadEq` z C $x{:}S\times T$

$\Gamma \vdash e_1 = e_2 \in x{:}S\times T$ ⌊ext Ax⌋

$\Gamma,\ s{:}S,\ t{:}T[s/x],\ y{:}\ e_1 = \langle s, t\rangle \in x{:}S\times T$

$\vdash t_1[s,t/x_1,y_1] = t_2[s,t/x_2,y_2] \in C[\langle s,t\rangle/z]$ ⌊ext Ax⌋

$\Gamma,\ z{:}\ x{:}S\times T,\ \Delta \vdash C$ ⌊ext $\mathsf{let}\ \langle s, t\rangle = z\ \mathsf{in}\ u$⌋

by `productElim` i

$\Gamma,\ z{:}\ x{:}S\times T,\ s{:}S,\ t{:}T[s/x]\ \Delta[\langle s,t\rangle/z]$

$\vdash C[\langle s,t\rangle/z]$ ⌊ext u⌋

$\Gamma \vdash \mathsf{let}\ \langle x, y\rangle = \langle s, t\rangle\ \mathsf{in}\ u = t_2 \in T$ ⌊ext Ax⌋

by `compute`

$\Gamma \vdash u[s,t/x,y] = t_2 \in T$ ⌊ext Ax⌋

Dependent Functions

Subsumes independent function type

$\forall$ generalizes $\Rightarrow$

Syntax:

Canonical: $x{:}S \to T$, $\lambda x\,.\,e$

Noncanonical: $e_1\ e_2$

Evaluation:

$$\frac{f \downarrow \lambda x.e' \quad e'[e/x] \downarrow val}{f\ \ e\ \downarrow\ val}$$

Semantics:

- $x{:}S \to T$ is a type if S is a type and $T[e/x]$ is a type for all e in S
- $\lambda x_1.\,e_1 = \lambda x_2.\,e_2$ in $x{:}S \to T$ if $x{:}S \to T$ type and
  $e_1[s_1/x_1] = e_2[s_2/x_2]$ in $T[s_1/x]$ for all s_1, s_2 with $s_1 = s_2 \in S$

See Appendix A.3.1 for further details