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 $q_0\in Q$, $\delta{:}Q\times\Sigma\to Q$, $F\subseteq Q$.
- **Represent dependent structures in programming languages**
  - Record types `[`f_1`:`T_1`; ...; `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, ...)

Dependent Products

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

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

$\Gamma \vdash S \times T_1 \;\text{type}$ $\quad\lfloor\text{ext } Ax\rfloor$
by $\times$-R
$\quad\Gamma \vdash S \;\text{type}$ $\quad\lfloor\text{ext } Ax\rfloor$
$\quad\Gamma \vdash T \;\text{type}$ $\quad\lfloor\text{ext } Ax\rfloor$

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

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

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

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

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` $\mathtt{M}_i$`(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 $\rightsquigarrow$ decidable typechecking
- **Typechecking in Nuprl cannot be fully automated**
  - Typechecking becomes part of the proof process $\rightsquigarrow$ heuristic typechecking
- **Additional problem**
  - What is the type of a function from $\mathbb{N}$ to types? $\rightsquigarrow$ Girard Paradox

Dependent Products: Further Inference rules

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

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

$\Gamma \vdash e_1 = e_2 \in x{:}S\times T$ $\lfloor\text{ext Ax}\rfloor$

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

$\vdash t_1[s,t/x_1,y_1] = t_2[s,t/x_2,y_2] \in C[\langle s,t\rangle/z]$ $\lfloor\text{ext Ax}\rfloor$

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

by `productElim` i

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

$\vdash C[\langle s,t\rangle/z]$ $\lfloor\text{ext } u\rfloor$

$\Gamma \vdash \text{let } \langle x, y\rangle = \langle s, t\rangle \text{ in } u = t_2 \in T$ $\lfloor\text{ext Ax}\rfloor$

by `compute`

$\Gamma \vdash u[s,t/x,y] = t_2 \in T$ $\lfloor\text{ext Ax}\rfloor$

Dependent Functions

Subsumes independent function type

$\forall$ generalizes $\Rightarrow$

Syntax:

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

Noncanonical: $e_1\;e_2$

Evaluation:

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

Semantics:

- $x{:}S{\rightarrow}T$ 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 $x{:}S{\rightarrow}T$ if $x{:}S{\rightarrow}T$ 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