CS576 Topics in Automated Deduction

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs576

Slides based in part on slides by Tobias Nipkow

March 5, 2015

Format for Inductive Relations Definitions

$$\texttt{inductive R :: ``}\tau \longrightarrow \texttt{bool''\ where}$$

$$[\![\texttt{R}(\texttt{a}_{1,1}); \ldots;\ \texttt{R}(\texttt{a}_{1,\texttt{n}});\ \texttt{A}_{1,1};\ \ldots;\ \texttt{A}_{1,\texttt{k}}]\!] \Longrightarrow \texttt{R}(\texttt{a}_1)\ \texttt{|}$$

$$\ldots\ \texttt{|}$$

$$[\![\texttt{R}(\texttt{a}_{\texttt{m},1}); \ldots;\ \texttt{R}(\texttt{a}_{\texttt{m},\texttt{l}});\ \texttt{A}_{\texttt{m},1};\ \ldots;\ \texttt{A}_{\texttt{m},\texttt{j}}]\!] \Longrightarrow \texttt{R}(\texttt{a}_\texttt{m})$$

where $\texttt{A}_{\texttt{i},\texttt{j}}$ are side conditions not involving $\texttt{R}$.

Format for Inductive Relations Definitions

$$\texttt{inductive R :: ``}\tau \longrightarrow \texttt{bool''\ where}$$
$$[\![\texttt{R}(\texttt{a}_{1,1}); \ldots; \texttt{R}(\texttt{a}_{1,\texttt{n}}); \texttt{A}_{1,1}; \ldots; \texttt{A}_{1,\texttt{k}}]\!] \Longrightarrow \texttt{R}(\texttt{a}_1) \mid$$
$$\ldots \mid$$
$$[\![\texttt{R}(\texttt{a}_{\texttt{m},1}); \ldots; \texttt{R}(\texttt{a}_{\texttt{m},\texttt{l}}); \texttt{A}_{\texttt{m},1}; \ldots; \texttt{A}_{\texttt{m},\texttt{j}}]\!] \Longrightarrow \texttt{R}(\texttt{a}_\texttt{m})$$

where $A_{i,j}$ are side conditions not involving R.

Format for Mutual Inductive Relations Definitions

$$
\begin{array}{l}
\texttt{inductive} \\
\mathtt{R_1} :: \text{“}\tau_1 \longrightarrow \texttt{bool}\text{”} \ \texttt{and} \\
\ldots \\
\mathtt{R_n} :: \text{“}\tau_n \longrightarrow \texttt{bool}\text{”} \ \texttt{where} \\
\quad [\![\mathtt{R_i}(\mathtt{a_{1,1}}); \ldots; \mathtt{R_j}(\mathtt{a_{1,n}}); \mathtt{A_{1,1}}; \ldots; \mathtt{A_{1,k}}]\!] \Longrightarrow \mathtt{R_k}(\mathtt{a_1}) \mid \\
\qquad \ldots \mid \\
\quad [\![\mathtt{R_m}(\mathtt{a_{m,1}}); \ldots; \mathtt{R_n}(\mathtt{a_{m,l}}); \mathtt{A_{m,1}}; \ldots; \mathtt{A_{m,j}}]\!] \Longrightarrow \mathtt{R_p}(\mathtt{a_m})
\end{array}
$$

where $\mathtt{A_{i,j}}$ are side conditions not involving any $\mathtt{R_k}$.

Example with Mutual Recursion

```
inductive
Even ::  "nat ⇒ bool" and
Odd ::  "nat ⇒ bool" where
ZeroEven [intro!]:  "Even 0" |
OddOne [intro!]:  "Odd (Suc 0)" |
OddSucEven [intro]:  "Odd n ⟹ Even (Suc n)" |
EvenSucOdd [intro]:  "Even n ⟹ Odd (Suc n)"
```

General Recursive Functions: `fun`

Example:

```
fun fib :: "nat ⇒ nat" where
    "fib 0 = 0" |
    "fib 1 = 1" |
    "fib (Suc(Suc x)) = (fib x + fib (Suc x))"
```

Not primitive recursive because of `fib(Suc(Suc x))` on left, and because of `fib(Suc x)` on right.

`fun`: Rules of Use

Compared to `primrec`, very few restrictions:

- Can be used to define functions over any type
- Clauses in `fun` must be equations
- Left-hand side is function being defined applied to terms built from data constructors, distinct variables and wildcards
- Right-hand side is a expression made from the function being defined, the variables in the argument on the left, and previously defined terms
- If clauses overlap, first takes precedence.
- Calculates a measure from lexicographic ordering of some collection of arguments

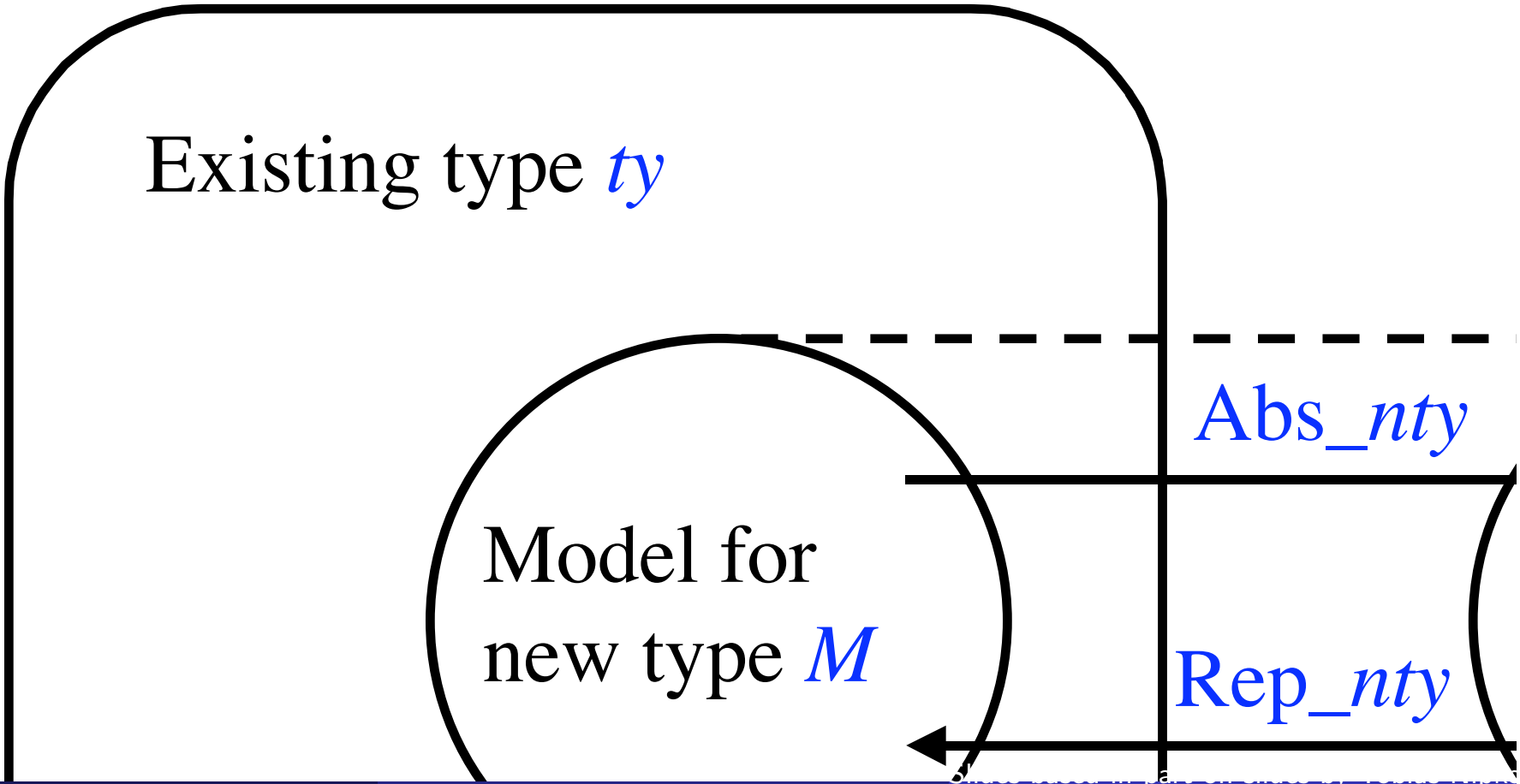
Example: sep

Define a function for putting a separator between all adjacent elements in a list:

```
fun sep ::  "'a * 'a list => 'a list" where
 "sep(a, []) = []" |
 "sep(a, [x]) = [x]" |
 "sep(a, x#y#zs) = x # a # sep(a,y#zs)"
```

Demo: General Recursive Functions

Introducing new types



Properties of a Defined Type Syntax for `typedef`:

$$\texttt{typedef}\ nty = \text{"}modeling\_set\text{"}$$
$$Proof\ of\ \exists x.x \in modeling\_set.$$

introduces a new type named *nty*, and functions

$$\texttt{Abs\_}nty :: ty \Rightarrow new\_ty$$
$$\texttt{Rep\_}nty :: new\_ty \Rightarrow ty$$

where *modeling_set*::*ty*

Properties of a Defined Type Theorems automatically provided:

Rep_*nty*: $\texttt{Rep\_}nty\ x \in modeling\_set$

Abs_*nty*_inverse: $\texttt{Abs\_}nty(\texttt{Rep\_}nty\ x) = x$

Rep_*nty*_inverse:

$y \in modeling\_set \Longrightarrow \texttt{Rep\_}nty(\texttt{Abs\_}nty\ y) = y$

Abs_*nty*_inject:

$[|x \in modeling\_set : y \in modeling\_set|] \Longrightarrow$

$(\texttt{Abs\_}nty\ x = \texttt{Abs\_}nty\ y) = (x = y)$

Rep_*nty*_inject: $(\texttt{Rep\_}nty\ x = \texttt{Rep\_}nty\ y) = (x = y)$