

Recursive Definitions

Wolfgang Schreiner
Engineering for Computer-based Learning
University of Applied Sciences at Hagenberg

Wolfgang.Schreiner@fh-hagenberg.at
<http://cbl.fh-hagenberg.at/~schreine>

Explicit Definitions

$$f(x_0, \dots, x_{n-1}) := T$$
$$p(x_0, \dots, x_{n-1}) :\Leftrightarrow F$$

- Definiendum (f, p) must not occur in definiens (T, F) .
- Restriction necessary to avoid contradictions:
 - **Invalid:** $f(x) := 1 + f(x)$.
 - **Invalid:** $p(x) :\Leftrightarrow \neg p(x)$.

Restriction may be lifted in certain cases.

Example

$$\begin{aligned} \mathit{fact}(n) &:= \\ &\mathbf{if} \ n = 0 \\ &\quad \mathbf{then} \ 1 \\ &\quad \mathbf{else} \ n * \mathit{fact}(n - 1) \end{aligned}$$
$$\begin{aligned} &\mathit{fact}(4) \\ &= 4 * \mathit{fact}(3) \\ &= 4 * (3 * \mathit{fact}(2)) \\ &= 4 * (3 * (2 * \mathit{fact}(1))) \\ &= 4 * (3 * (2 * (1 * \mathit{fact}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) = 24 \end{aligned}$$

Well-defined function $\mathit{fact} : \mathbb{N} \rightarrow \mathbb{N}$.

Example

$$\begin{aligned} \text{foo}(n) &:= \\ &\mathbf{if} \ n = 0 \\ &\quad \mathbf{then} \ 1 \\ &\quad \mathbf{else} \ n * \text{foo}(n + 1) \end{aligned}$$
$$\begin{aligned} &\text{foo}(4) \\ &= 4 * \text{foo}(5) \\ &= 4 * (5 * \text{foo}(6)) \\ &= 4 * (5 * (6 * \text{foo}(7))) \\ &= \dots \end{aligned}$$

Not a well-defined function.

Well-Founded Ordering

When is a recursive definition well-behaved?

- Two conditions:

1. Argument must get “smaller” in every recursive invocation.

- $fact(n) := n * fact(n - 1)$, for $n > 0$.

2. Function value for the “smallest” argument is defined without recursion.

- $fact(0) := 1$.

- **Well-founded ordering:**

- A binary relation \prec is well-founded if there is no infinitely decreasing chain w.r.t. \prec :

- \prec is well-founded $:\Leftrightarrow \neg \exists S, s : \mathbb{N} \rightarrow S : \forall i \in \mathbb{N} : s_{i+1} \prec s_i$.

- The binary relation $<$ on \mathbb{N} is well-founded.

- $0 < \dots < n - 2 < n - 1 < n$.

Recursive Function Definitions

$$f(x_0, \dots, x_{n-1}) := T$$

- Recursive function definition is well-formed if
 - there exists a well-founded ordering \prec on the function domain such that for all arguments x_0, \dots, x_{n-1} and for every occurrence of $f(T_0, \dots, T_{n-1})$ in T needed to compute $f(x_0, \dots, x_{n-1})$, we have

$$\langle T_0, \dots, T_{n-1} \rangle \prec \langle x_0, \dots, x_{n-1} \rangle.$$

- Definition introduces an n -ary function constant f such that

$$\forall x_0, \dots, x_{n-1} : f(x_0, \dots, x_{n-1}) = T.$$

We must make sure that such a well-founded ordering exists.

Structure of Recursive Function Definitions

Which terms are “needed” to compute a function result?

- Every subterm of T is needed to determine the value of T , **unless** T is of the form

if F then T_0 else T_1 .

- Only the value of T_0 is needed if F is true, and only the value of T_1 is needed, otherwise.
- Consequence: the following definition **cannot** be well-formed.

$$f(x) := 1 + f(x - 1)$$

- Every subterm is needed to compute the function result.
- $f(0)$ cannot be computed.

Every recursive function definition must have a condition in the body.

Termination Function

How can we ensure that the right well-founded ordering exists?

- Define a **termination function**.
 - Recursive function definition $f : A \rightarrow B$.
 - Termination function $r : A \rightarrow \mathbb{N}$.
- Well-founded ordering:

$$x \prec y :\Leftrightarrow r(x) < r(y)$$

Use the ordering of the natural numbers.

Example

$$* : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$x * y :=$$

$$\mathbf{if} \ y = 0$$

$$\mathbf{then} \ 0$$

$$\mathbf{else} \ x + x * (y - 1)$$

$$\begin{aligned} \underline{2 * 3} &= 2 + (\underline{2 * 2}) = 2 + (2 + \underline{2 * 1}) = \\ &2 + (2 + (2 + (\underline{2 * 0}))) = 2 + (2 + (2 + 0)) = 6. \end{aligned}$$

- Termination function $r(x, y) := y$.

- If $r(x, y) = 0$ (i.e., $y = 0$), no recursive application is required to determine $x * y$.
- If $r(x, y) \neq 0$ (i.e., $y \neq 0$), we need $x * (y - 1)$ with $r(x, y - 1) = y - 1 < y = r(x, y)$.

Example

Bogus recursive definition.

$$* : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$x * y :=$$

$$\mathbf{if} \ y = 0$$

$$\mathbf{then} \ 0$$

$$\mathbf{else} \ x + x * (y + 1)$$

$$\underline{2 * 3} = 2 + (\underline{2 * 4}) = 2 + (2 + \underline{2 * 5}) = \\ 2 + (2 + (2 + (\underline{2 * 6}))) = \dots$$

No termination function can be given.

Example

$$\text{sum} : \text{FiniteSet} \rightarrow \mathbb{N}$$

$$\text{sum}(S) :=$$

$$\quad \mathbf{if} \ S = \emptyset$$

$$\quad \mathbf{then} \ 0$$

$$\quad \mathbf{else} \ \text{any}(S) + \text{sum}(\text{rest}(S))$$

$$\text{any}(S) := \mathbf{such} \ x : x \in S$$

$$\text{rest}(S) := \{x \in S : x \neq \text{any}(S)\}$$

- Termination function $r(S) := |S|$.
 - If $r(S) = 0$ (i.e., $S = \emptyset$), no recursive application is needed to compute $\text{sum}(S)$.
 - If $r(S) \neq 0$ (i.e. $S \neq \emptyset$), we need $\text{sum}(\text{rest}(S))$ with $r(\text{rest}(S)) = r(S) - 1 < r(S)$.

Example

$$\begin{aligned} \text{dsum} &: \text{FiniteSet} \times \text{FiniteSet} \rightarrow \mathbb{N} \\ \text{dsum}(A, B) &:= \\ &\quad \mathbf{if} \ A = \emptyset \ \wedge \ B = \emptyset \ \mathbf{then} \\ &\quad \quad 0 \\ &\quad \mathbf{else if} \ A = \emptyset \ \mathbf{then} \\ &\quad \quad \text{any}(B) + \text{dsum}(A, \text{rest}(B)) \\ &\quad \mathbf{else if} \ B = \emptyset \ \mathbf{then} \\ &\quad \quad \text{any}(A) + \text{dsum}(\text{rest}(A), B) \\ &\quad \mathbf{else} \\ &\quad \quad \text{any}(A) + \text{any}(B) + \text{dsum}(\text{rest}(A), \text{rest}(B)). \end{aligned}$$

- Termination function $r(A, B) := |A| + |B|$.
 - If $r(A, B) = 0$ (i.e., $A \cup B = \emptyset$), no recursive application is needed to compute $\text{dsum}(A, B)$.
 - If $r(A, B) \neq 0$ (i.e. $A \cup B \neq \emptyset$), we need
 - * $\text{dsum}(A, \text{rest}(B))$ (with $r(A, \text{rest}(B)) = r(A, B) - 1 < r(A, B)$) or
 - * $\text{dsum}(\text{rest}(A), B)$ (with $r(\text{rest}(A), B) = r(A, B) - 1 < r(A, B)$) or
 - * $\text{dsum}(\text{rest}(A), \text{rest}(B))$ (with $r(\text{rest}(A), \text{rest}(B)) = r(A, B) - 2 < r(A, B)$).

Logic Evaluator

```
fun  $f(x_0, \dots, x_{n-1})$  recursive  $R = T$ ;
```

```
option silent = true; read set;
```

```
fun *(x: Nat, y: Nat) recursive y =  
  if(=(y, 0), 0, +(x, *(x, -(y, 1))));
```

```
fun any(S: Set) = such(x in S: true, x);  
fun rest(S: Set) = let(e = any(S): set(x in S: not(=(e, x)), x));
```

```
fun size(S: Set) recursive #(S) =  
  if(=(S, {}), 0, +(1, size(rest(S))));
```

```
fun prod(S: Set) recursive #(S) =  
  if(=(S, {}), 1, *(any(S), prod(rest(S))));
```

Recursive Predicate Definitions

$$p(x_0, \dots, x_{n-1}) :\Leftrightarrow F$$

- Recursive predicate definition is well-formed if
 - there exists a well-founded ordering \prec on the predicate domain such that for all arguments x_0, \dots, x_{n-1} and for every occurrence of $p(T_0, \dots, T_{n-1})$ in T needed to compute $p(x_0, \dots, x_{n-1})$, we have

$$\langle T_0, \dots, T_{n-1} \rangle \prec \langle x_0, \dots, x_{n-1} \rangle.$$

- Definition introduces an n -ary predicate constant p such that

$$\forall x_0, \dots, x_{n-1} : p(x_0, \dots, x_{n-1}) \Leftrightarrow F.$$

We must make sure that such a well-founded ordering exists.

Example

$$\text{iseven} \subseteq \mathbb{N}$$

$$\text{iseven}(x) :\Leftrightarrow$$

$$\mathbf{if} \ x = 0$$

$$\mathbf{then} \ T$$

$$\mathbf{else} \ \neg \text{iseven}(x - 1)$$

$$\text{iseven}(3) = \neg \text{iseven}(2) = \neg \neg \text{iseven}(1) = \neg \neg \neg \text{iseven}(0) =$$

$$\neg \neg \neg T = \neg \neg F = \neg T = F.$$

- Termination function $r(x) := x$.
 - If $r(x) = 0$ (i.e., $x = 0$), no recursive invocation is needed.
 - If $r(x) \neq 0$ (i.e., $x \neq 0$), the value of $\text{iseven}(x - 1)$ (with $r(x - 1) = x - 1 < r(x)$) is needed.

Example

$$\leq \subseteq \mathbb{N} \times \mathbb{N}$$

$$x \leq y :\Leftrightarrow$$

if $x = 0$ **then** T

else if $y = 0$ **then** F

else $x - 1 \leq y - 1$

$$3 \leq 4 \Leftrightarrow 2 \leq 3 \Leftrightarrow 1 \leq 2 \Leftrightarrow 0 \leq 1 \Leftrightarrow \text{T.}$$

- Termination function $r(x, y) := x$.
 - If $r(x, y) = 0$ (i.e., $x = 0$), no recursive invocation is needed.
 - If $r(x, y) \neq 0$ (i.e., $x \neq 0$), the value of $x - 1 \leq y - 1$ (with $r(x - 1, y - 1) = x - 1 < r(x, y)$) is needed.

Logic Evaluator

```
pred  $p(x_0, \dots, x_{n-1})$  recursive  $R \Leftrightarrow F$ ;
```

```
pred iseven(x: Nat) recursive x  $\Leftrightarrow$   
  if(=(x, 0), true, not(iseven(-(x, 1))));
```

```
pred <=(x: Nat, y: Nat) recursive x  $\Leftrightarrow$   
  if(=(x, 0), true,  
    if(=(y, 0), false,  
      <=(-(x, 1), -(y, 1))));
```

Reduction of Sets

Functions over sets are frequently defined as follows.

$$g(S) :=$$

if $S = \emptyset$ **then** b
else let $e = \text{such } x : x \in S :$
 $f(e, g(S - \{e\}))$.

- **Reduction** of S by f with base b .
- If f is commutative and associative, the result of a reduction is uniquely defined.

Recursive function definition pattern.

Example

Let $S := \{1, 2, 3\}$.

Then the reduction of S by $+$ with base 0 is

$$6 = (1 + (2 + 3)) + 0 = (2 + (1 + 3)) + 0 = (2 + 1) + (3 + 0).$$

The reduction of S by $*$ with base 1 is

$$6 = (1 * (2 * 3)) * 1 = (2 * (1 * 3)) * 1 = ((1 * 1) * 2) * 3.$$

Many interesting set functions can be defined by reduction.

Logic Evaluator

```
reduce(f, S, b)
```

```
fun sum(S) = reduce(+, S, 0);
```

```
fun product(S) = reduce(*, S, 1);
```

```
fun +1(x, y) = +(1, y);
```

```
fun #(S) = reduce(+1, S, 0);
```

```
fun sum'(S) recursive #(S) =
```

```
  if(=(S, {}), 0,
```

```
    let(e = such(x in S: true, x):
```

```
      +(e, sum'(set(x in S: not(=(x, e)), x)))));
```

Set Union and Powerset

```
fun ++(A: Set, B: Set) =  
  reduce(join, A, B);
```

```
fun combine(e, S: Set) =  
  ++(S, set(x in S: true, join(e, x)));
```

```
fun Powerset(S: Set) =  
  reduce(combine, S, join({}, {}));
```