Appendix A

Selected ACL2 functions and macros

Given below are logical definitions of some of the primitive ACL2 functions which are used in the Appendix C.1 to formalise event reconstruction. Please refer to [49] for more detail.

A.1 Functions

A.1.1 Logical functions

$$\texttt{(equal x y)} = \begin{cases} \texttt{t} & \text{if } \texttt{x} = \texttt{y} \\ \texttt{nil} & \text{if } \texttt{x} \neq \texttt{y} \end{cases}$$

$$\texttt{(if x y z)} = \begin{cases} \texttt{y} & \text{if } \texttt{x} \neq \texttt{nil} \\ \texttt{z} & \text{if } \texttt{x} = \texttt{nil} \end{cases}$$

$$\texttt{(not x)} = \texttt{(if x nil t)}$$

$$\texttt{(implies x y)} = \texttt{(if x (if y t nil) t))}$$

$$\texttt{(iff x y)} = \texttt{(if x (if y t nil) (if y nil t)))}$$

A.1.2 Integer functions

$$\texttt{(integerp x)} = \begin{cases} \texttt{t} & \text{if } \texttt{x} \text{ is an integer} \\ \texttt{nil} & \text{if } \texttt{x} \text{ is not an integer} \end{cases}$$

$$\texttt{(ifix x)} = \begin{cases} \texttt{x} & \text{if } \texttt{x} \text{ is an integer} \\ 0 & \text{if } \texttt{x} \text{ is not an integer} \end{cases}$$

$$\texttt{(nfix x)} = \begin{cases} \texttt{x} & \text{if } \textit{integerp}(\texttt{x}) \text{ and } \texttt{x} > 0 \\ 0 & \text{if } \neg\textit{integerp}(\texttt{x}) \text{ or } \texttt{x} <= 0 \end{cases}$$

$$\texttt{(1+ x)} = \texttt{(ifix x)} + 1$$

$$\texttt{(1- x)} = \texttt{(ifix x)} - 1$$

$$\texttt{(binary-+ x y)} = \texttt{(ifix x)} + \texttt{(ifix y)}$$

$$\texttt{(binary-* x y)} = \texttt{(ifix x)} * \texttt{(ifix y)}$$

$$\texttt{(unary-- x)} = -\texttt{(ifix x)}$$

$$\texttt{(< x y)} = \begin{cases} \texttt{t} & \text{if } \texttt{(ifix x)} < \texttt{(ifix y)} \\ \texttt{nil} & \text{if } \texttt{(ifix x)} \geq \texttt{(ifix y)} \end{cases}$$

A.1.3 Functions for manipulating ordered pairs

$$\texttt{(consp x)} = \begin{cases} \texttt{t} & \text{if } \texttt{x} \text{ is an ordered pair} \\ \texttt{nil} & \text{if } \texttt{x} \text{ is not an ordered pair} \end{cases}$$

$$\texttt{(atom x)} = \texttt{(not (consp x))}$$

$\texttt{(cons x y)}$ = ordered pair whose first element is $\texttt{x}$ and second element is $\texttt{y}$

$$\texttt{(car x)} = \begin{cases} \text{first element of x} & \text{if } \texttt{x} \text{ is an ordered pair} \\ \texttt{nil} & \text{if } \texttt{x} \text{ is not an ordered pair} \end{cases}$$

$$\texttt{(cdr x)} = \begin{cases} \text{second element of x} & \text{if } \texttt{x} \text{ is an ordered pair} \\ \texttt{nil} & \text{if } \texttt{x} \text{ is not an ordered pair} \end{cases}$$

A.1.4 Functions for manipulating lists

```
(defun len (x) (if (consp x) (+ 1 (len (cdr x))) 0)
```

```
(defun binary-append (x y)
   (if (consp x)
     (cons (car x) (binary-append (cdr x) y))
     y))
```

```
(defun con (x)
   (if (consp x)
     (binary-append (car x) (con (cdr x)))
     nil))
```

A.2 Macros

- `(and a1 a2 ... an)`. This expands to

  ```
  (if a1 (if a2 ... (if an t nil) ... nil) nil)
  ```

  Logically this is equivalent to $\texttt{a1} \wedge \texttt{a1} \wedge \ldots \wedge \texttt{an}$.

- `(or a1 a2 ... an)`.

  ```
  (if a1 t (if a1 t ... (if an t nil) ...))
  ```

  Logically this is equivalent to $\texttt{a1} \vee \texttt{a1} \vee \ldots \vee \texttt{an}$.

- `(+ a1 a2 ... an-1 an)`. This expands to

  ```
  (binary-+ a1 (binary-+ a2 ... (binary-+ an-1 an) ... ))
  ```

  If $\texttt{a1}, \texttt{a2}, \ldots, \texttt{an}$ are numbers, this is equivalent to $\texttt{a1} + \texttt{a2} + \ldots + \texttt{an}$

- `(- a b)` expands to `(binary-+ a (unary-- b))`

- `(let bindings body)`, is equivalent to the result of substituting terms from `bindings` for corresponding variables in the `body`.

  For example, `(let ((a (+ y z))) (equal a b))` expands to

  `(equal (+ y z) b)`.