CS 4301/6371	**Advanced Programming Languages**	Sample
Spring 2024		Midterm Exam

Write your name at the top of this exam paper and turn it in as the front page of your submission. You may take a single, two-sided sheet of notes with you into the exam. All other books or notes must remain closed throughout the exam. You will have the duration of the class period to complete the exam; all papers must be turned in by 2:15pm. Good luck!

(1) A number can be encoded as a list of digits. For example, the list `[1;0;2;4]` encodes the number 1024 in base-10, and the list `[15;15]` encodes 0xFF=255 in base-16. In this problem you will implement OCaml functions that convert between integers and digit lists. Do not use any `List` library functions in your solutions except that you may use `List.fold_left` if you wish. In case you need them, the OCaml integer-division and integer-modulo operators are `(x/y)` and `(x mod y)`, respectively.

(a) (7 pts) Implement a **tail-recursive** OCaml function `(digitize b n)` that converts an integer `n` into a base-`b` digit list. You may assume that $\texttt{b} \geq 2$ and $\texttt{n} \geq 0$.

(b) (6 pts) Implement a **tail-recursive** OCaml function `(undigitize b dl)` that converts a base-`b` digit list `dl` to an integer. You may assume that $\texttt{b} \geq 2$ and that no elements of `dl` are less than 0 or greater than $\texttt{b} - 1$.

(2) (14 pts) Recall following definition of `map` from class:

```
let rec map f = function [] -> [] | h::t -> (f h)::(map f t);;
```

Unfortunately this definition is not tail-recursive. Explain why. What is the impact in terms of time and space? Fix this problem by finding a new implementation of `map` that uses only `fold_left` for recursion.

(3) Consider the following extension to SIMPL, which defines the syntax and large-step semantics of a new arithmetic expression called `calc`:

$$a ::= \cdots \mid \texttt{calc}(v, a_1, a_2)$$

$$\frac{\langle a_1, \sigma\rangle \Downarrow n_1 \qquad \langle a_2, \sigma\rangle \Downarrow n_2 \qquad n_1 \leq n_2}{\langle \texttt{calc}(v, a_1, a_2), \sigma\rangle \Downarrow \sigma(v)} \tag{C1}$$

$$\frac{\langle a_1, \sigma\rangle \Downarrow n_1 \qquad \langle a_2, \sigma\rangle \Downarrow n_2 \qquad n_1 > n_2 \qquad \langle \texttt{calc}(v, a_1, a_2), \sigma[v \mapsto \sigma(v)+1]\rangle \Downarrow n}{\langle \texttt{calc}(v, a_1, a_2), \sigma\rangle \Downarrow n} \tag{C2}$$

(a) (8 pts) Define $\sigma = \{(\texttt{x}, 1)\}$. Derive $\langle \texttt{calc}(\texttt{x}, \texttt{x}+1, \texttt{x}*\texttt{x}), \sigma\rangle \Downarrow n$ for any integer n.

(b) (5 pts) Write a "plain English" description of $\texttt{calc}(v, a_1, a_2)$ for a programmer's reference manual. Your description should not refer to inference rules, stores, or derivations since most programmers aren't familiar with such concepts. Is it possible for `calc` to ever loop infinitely? Your description should either explain under what conditions it can, or why it cannot. What is the final value of v after the expression finishes evaluating?

(c) (10 pts) Write small-step operational semantic rules for `calc` that are equivalent to the large-step rules given above. (You need not prove semantic equivalence.)

(4) Consider the following recursive definition of function f:

$$f(x) = \big(x{=}100 \to 0 \mid x{>}100 \to f(x-1)+1 \mid x{<}100 \to f(x+1)+1\big)$$

(a) (1 pt) Define a non-recursive functional F whose least fixed point is f.

(b) (4 pts) Give a closed-form function definition h such that $h = f$.

(c) (15 pts) Prove by fixed point induction that $h = \mathit{fix}(F)$.

Solutions

(1) **(a)**
```
let digitize b n =
    let rec f dl n = if n<=0 then dl else f ((n mod b)::dl) (n/b)
    in f [] n;;
```

(b)
```
let undigitize b = fold_left (fun a d -> a*b+d) 0;;
```

(2) The implementation is not tail-recursive because the recursive call does not directly return its result; it pushes `(f h)` onto the returned list, requiring the caller's stack frame to persist. This causes the program to use $O(n)$ stack space, where n is the length of the input list. (There is no impact on running time, save possibly for the results of poor locality.)

To fix the problem, we could implement the algorithm in two passes: The first pass reverses the list as it iterates, thereby avoiding a copy of the list pointers on the stack; and the second reverses it back. Both passes are tail-recursive, so the implementation is tail-recursive.

```
let map f l =
  fold_left (fun t h -> h::t) [] (fold_left (fun t h -> (f h)::t) [] l);;
```

(3) **(a)** The correct integer is $n = 2$:

$$\dfrac{\dfrac{\mathcal{D}_1 \quad \dfrac{}{\langle 1, \sigma\rangle \Downarrow 1}{\scriptstyle(13)}}{\langle \mathtt{x+1}, \sigma\rangle \Downarrow 2}{\scriptstyle(15)} \quad \dfrac{\mathcal{D}_1 \quad \mathcal{D}_1}{\langle \mathtt{x*x}, \sigma\rangle \Downarrow 1}{\scriptstyle(17)} \quad 2 > 1 \quad \dfrac{\mathcal{D}_2 \quad \dfrac{}{\langle 1, \sigma'\rangle \Downarrow 1}{\scriptstyle(13)}}{\langle \mathtt{x+1}, \sigma'\rangle \Downarrow 3}{\scriptstyle(15)} \quad \dfrac{\mathcal{D}_2 \quad \mathcal{D}_2}{\langle \mathtt{x*x}, \sigma'\rangle \Downarrow 4}{\scriptstyle(17)} \quad 3 \le 4}{\langle \mathtt{calc(x, x+1, x*x)}, \sigma'\rangle \Downarrow 2}{\scriptstyle(C1)}}{\langle \mathtt{calc(x, x+1, x*x)}, \sigma\rangle \Downarrow 2}{\scriptstyle(C2)}$$

where $\sigma' = \sigma[\mathtt{x} \mapsto 2]$, $\mathcal{D}_1 = \dfrac{}{\langle \mathtt{x}, \sigma\rangle \Downarrow 1}{\scriptstyle(14)}$, and $\mathcal{D}_2 = \dfrac{}{\langle \mathtt{x}, \sigma'\rangle \Downarrow 2}{\scriptstyle(14)}$.

(b) Expression $\mathtt{calc}(v, a_1, a_2)$ returns the smallest integer no less than v that satisfies $a_1 \le a_2$ when substituted for v in a_1 and a_2. If there is no integer satisfying these constraints, the result is implementation-defined (possibly an infinite loop). The value of v is *not changed* by evaluating the expression.

(c) To formulate this as small-step semantics, we need some new kind of configuration that tracks the progress of the `calc` loop as it iterates. For example, the following rules use $\langle \mathtt{calc}_{n,a_1,a_2}(\ldots), \sigma\rangle$ as an intermediate state.

$$\frac{\sigma(v) = n}{\langle \mathtt{calc}(v, a_1, a_2), \sigma\rangle \to_1 \langle \mathtt{calc}_{n,a_1,a_2}(v, a_1, a_2), \sigma\rangle} \tag{SC1}$$

$$\frac{\langle a_1', \sigma\rangle \to_1 \langle a_1'', \sigma'\rangle}{\langle \mathtt{calc}_{n,a_1,a_2}(v, a_1', a_2'), \sigma\rangle \to_1 \langle \mathtt{calc}_{n,a_1,a_2}(v, a_1'', a_2'), \sigma'\rangle} \tag{SC2}$$

$$\frac{\langle a_2', \sigma\rangle \to_1 \langle a_2'', \sigma'\rangle}{\langle \mathtt{calc}_{n,a_1,a_2}(v, n_1, a_2'), \sigma\rangle \to_1 \langle \mathtt{calc}_{n,a_1,a_2}(v, n_1, a_2''), \sigma'\rangle} \tag{SC3}$$

$$\frac{n_1 \le n_2}{\langle \mathtt{calc}_{n,a_1,a_2}(v, n_1, n_2), \sigma\rangle \to_1 \langle \sigma(v), \sigma[v \mapsto n]\rangle} \tag{SC4}$$

$$\frac{n_1 > n_2}{\langle \mathtt{calc}_{n,a_1,a_2}(v, n_1, n_2), \sigma\rangle \to_1 \langle \mathtt{calc}_{n,a_1,a_2}(v, a_1, a_2), \sigma[v \mapsto \sigma(v) + 1]\rangle} \tag{SC5}$$

(4) **(a)** $F(g) = \lambda x.\big(x{=}100 \to 0 \mid x{>}100 \to g(x-1)+1 \mid x{<}100 \to g(x+1)+1\big)$

(b) $h(x) = |x - 100|$

(c) *Proof.* Define proposition $P(g) = \forall x \in g^{\leftarrow} \,.\, g(x) = |x - 100|$. We wish to prove that $P(\mathit{fix}(F))$ holds.

Base Case: $P(\bot)$ holds vacuously.

Inductive Case: As the inductive hypothesis, assume that $P(g)$ holds. We must prove that $P(F(g))$ holds. Let $x \in F(g)^{\leftarrow}$ be given.

Case 1: Suppose $x = 100$. Then by definition of F, $F(g)(x) = 0 = |x - 100|$.

Case 2: Suppose $x > 100$. Then by definition of F, $F(g)(x) = g(x-1)+1$. By inductive hypothesis, $g(x-1) = |x-1-100|$. Since $x > 100$, $|x-1-100|+1 = x-1-100+1 = x-100 = |x-100|$.

Case 3: Suppose $x < 100$. Then by definition of F, $F(g)(x) = g(x+1)+1$. Since $x < 100$, $|x+1-100|+1 = -(x+1-100)+1 = -(x-100) = |x-100|$. $\square$

Reference

Here is the syntax, large- and small-step operational semantics, and denotational semantics of SIMPL that were defined in class. These definitions will be provided to you with your exam.

Syntax of SIMPL

commands	$c ::= \texttt{skip} \mid c_1; c_2 \mid v\texttt{:=}a \mid \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \mid \texttt{while } b \texttt{ do } c$
boolean expressions	$b ::= \texttt{true} \mid \texttt{false} \mid a_1\texttt{<=}a_2 \mid b_1 \texttt{ \&\& } b_2 \mid b_1 \texttt{ || } b_2 \mid \texttt{!}b$
arithmetic expressions	$a ::= n \mid v \mid a_1 \texttt{ + } a_2 \mid a_1 \texttt{ - } a_2 \mid a_1 \texttt{ * } a_2$
variable names	v
integer constants	n

Large-step Semantics of SIMPL

Commands

$$\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma \tag{L1}$$

$$\frac{\langle c_1, \sigma \rangle \Downarrow \sigma_2 \qquad \langle c_2, \sigma_2 \rangle \Downarrow \sigma'}{\langle c_1; c_2, \sigma \rangle \Downarrow \sigma'} \tag{L2}$$

$$\frac{\langle a, \sigma \rangle \Downarrow n}{\langle v\texttt{:=}a, \sigma \rangle \Downarrow \sigma[v \mapsto n]} \tag{L3}$$

$$\frac{\langle b, \sigma \rangle \Downarrow T \qquad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \Downarrow \sigma'} \tag{L4}$$

$$\frac{\langle b, \sigma \rangle \Downarrow F \qquad \langle c_2, \sigma \rangle \Downarrow \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \Downarrow \sigma'} \tag{L5}$$

$$\frac{\langle \texttt{if } b \texttt{ then } (c; \texttt{while } b \texttt{ do } c) \texttt{ else skip}, \sigma \rangle \Downarrow \sigma'}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \Downarrow \sigma'} \tag{L6}$$

Boolean Expressions

$$\langle \texttt{true}, \sigma \rangle \Downarrow T \tag{L7}$$

$$\langle \texttt{false}, \sigma \rangle \Downarrow F \tag{L8}$$

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1\texttt{<=}a_2, \sigma \rangle \Downarrow n_1 \leq n_2} \tag{L9}$$

$$\frac{\langle b_1, \sigma \rangle \Downarrow p \qquad \langle b_2, \sigma \rangle \Downarrow q}{\langle b_1 \texttt{ \&\& } b_2, \sigma \rangle \Downarrow p \wedge q} \tag{L10}$$

$$\frac{\langle b_1, \sigma \rangle \Downarrow p \qquad \langle b_2, \sigma \rangle \Downarrow q}{\langle b_1 \texttt{ || } b_2, \sigma \rangle \Downarrow p \vee q} \tag{L11}$$

$$\frac{\langle b, \sigma \rangle \Downarrow p}{\langle \texttt{!}b, \sigma \rangle \Downarrow \neg p} \tag{L12}$$

Arithmetic Expressions

$$\langle n, \sigma \rangle \Downarrow n \tag{L13}$$

$$\langle v, \sigma \rangle \Downarrow \sigma(v) \tag{L14}$$

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 \texttt{ + } a_2, \sigma \rangle \Downarrow n_1 + n_2} \tag{L15}$$

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 \texttt{ - } a_2, \sigma \rangle \Downarrow n_1 - n_2} \tag{L16}$$

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 \texttt{ * } a_2, \sigma \rangle \Downarrow n_1 n_2} \tag{L17}$$

Small-step Semantics of SIMPL

Commands

$$\frac{\langle c_1, \sigma\rangle \to_1 \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \to_1 \langle c_1'; c_2, \sigma'\rangle} \tag{S1}$$

$$\langle \texttt{skip}; c_2, \sigma\rangle \to_1 \langle c_2, \sigma\rangle \tag{S2}$$

$$\frac{\langle a, \sigma\rangle \to_1 \langle a', \sigma'\rangle}{\langle v\,\texttt{:=}\,a, \sigma\rangle \to_1 \langle v\,\texttt{:=}\,a', \sigma'\rangle} \tag{S3}$$

$$\langle v\,\texttt{:=}\,n, \sigma\rangle \to_1 \langle \texttt{skip}, \sigma[v \mapsto n]\rangle \tag{S4}$$

$$\frac{\langle b, \sigma\rangle \to_1 \langle b', \sigma'\rangle}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \to_1 \langle \texttt{if}\ b'\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma'\rangle} \tag{S5}$$

$$\langle \texttt{if true then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \to_1 \langle c_1, \sigma\rangle \tag{S6}$$

$$\langle \texttt{if false then}\ c_1\ \texttt{else}\ c_2, \sigma\rangle \to_1 \langle c_2, \sigma\rangle \tag{S7}$$

$$\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \to_1 \langle \texttt{if}\ b\ \texttt{then}\ (c; \texttt{while}\ b\ \texttt{do}\ c)\ \texttt{else skip}, \sigma\rangle \tag{S8}$$

Boolean Expressions

$$\frac{\langle a_1, \sigma\rangle \to_1 \langle a_1', \sigma'\rangle}{\langle a_1 \texttt{<=} a_2, \sigma\rangle \to_1 \langle a_1' \texttt{<=} a_2, \sigma'\rangle} \tag{S9}$$

$$\frac{\langle a_2, \sigma\rangle \to_1 \langle a_2', \sigma'\rangle}{\langle n_1 \texttt{<=} a_2, \sigma\rangle \to_1 \langle n_1 \texttt{<=} a_2', \sigma'\rangle} \tag{S10}$$

$$\frac{n_1 \le n_2}{\langle n_1 \texttt{<=} n_2, \sigma\rangle \to_1 \langle \texttt{true}, \sigma\rangle} \tag{S11}$$

$$\frac{n_1 > n_2}{\langle n_1 \texttt{<=} n_2, \sigma\rangle \to_1 \langle \texttt{false}, \sigma\rangle} \tag{S12}$$

$$\frac{\langle b_1, \sigma\rangle \to_1 \langle b_1', \sigma'\rangle \quad op \in \{\,\texttt{\&\&}\,,\ \texttt{||}\,\}}{\langle b_1\ op\ b_2, \sigma\rangle \to_1 \langle b_1'\ op\ b_2, \sigma'\rangle} \tag{S13}$$

$$\langle \texttt{true \&\& } b_2, \sigma\rangle \to_1 \langle b_2, \sigma\rangle \tag{S14}$$

$$\langle \texttt{false \&\& } b_2, \sigma\rangle \to_1 \langle \texttt{false}, \sigma\rangle \tag{S15}$$

$$\langle \texttt{true || } b_2, \sigma\rangle \to_1 \langle \texttt{true}, \sigma\rangle \tag{S16}$$

$$\langle \texttt{false || } b_2, \sigma\rangle \to_1 \langle b_2, \sigma\rangle \tag{S17}$$

$$\frac{\langle b, \sigma\rangle \to_1 \langle b', \sigma'\rangle}{\langle \texttt{!}b, \sigma\rangle \to_1 \langle \texttt{!}b', \sigma'\rangle} \tag{S18}$$

$$\langle \texttt{!true}, \sigma\rangle \to_1 \langle \texttt{false}, \sigma\rangle \tag{S19}$$

$$\langle \texttt{!false}, \sigma\rangle \to_1 \langle \texttt{true}, \sigma\rangle \tag{S20}$$

Arithmetic Expressions

$$\langle v, \sigma\rangle \to_1 \langle \sigma(v), \sigma\rangle \tag{S21}$$

$$\frac{\langle a_1, \sigma\rangle \to_1 \langle a_1', \sigma'\rangle \qquad op \in \{\,\texttt{+}\,,\ \texttt{-}\,,\ \texttt{*}\,\}}{\langle a_1\ op\ a_2, \sigma\rangle \to_1 \langle a_1'\ op\ a_2, \sigma'\rangle} \tag{S22}$$

$$\frac{\langle a_2, \sigma\rangle \to_1 \langle a_2', \sigma'\rangle \qquad op \in \{\,\texttt{+}\,,\ \texttt{-}\,,\ \texttt{*}\,\}}{\langle n_1\ op\ a_2, \sigma\rangle \to_1 \langle n_1\ op\ a_2', \sigma'\rangle} \tag{S23}$$

$$\langle n_1 \texttt{ + } n_2, \sigma\rangle \to_1 \langle n_1 + n_2, \sigma\rangle \tag{S24}$$

$$\langle n_1 \texttt{ - } n_2, \sigma\rangle \to_1 \langle n_1 - n_2, \sigma\rangle \tag{S25}$$

$$\langle n_1 \texttt{ * } n_2, \sigma\rangle \to_1 \langle n_1 n_2, \sigma\rangle \tag{S26}$$

Denotational Semantics

$$\begin{aligned}
&\Sigma = v \rightharpoonup \mathbb{Z} \\
&\mathcal{A} : a \to (\Sigma \rightharpoonup \mathbb{Z}) \\
&\mathcal{B} : b \to (\Sigma \rightharpoonup \{T, F\}) \\
&\mathcal{C} : c \to (\Sigma \rightharpoonup \Sigma)
\end{aligned}$$

Arithmetic Expressions

$$\mathcal{A}[\![n]\!] = \{(\sigma, n) \mid \sigma \in \Sigma\} \tag{D1}$$

$$\mathcal{A}[\![x]\!] = \{(\sigma, \sigma(x)) \mid \sigma \in \Sigma,\ x \in \sigma^{\leftarrow}\} \tag{D2}$$

$$\mathcal{A}[\![a_1 \texttt{+} a_2]\!] = \{(\sigma, n_1 + n_2) \mid n_1 = \mathcal{A}[\![a_1]\!]\sigma,\ n_2 = \mathcal{A}[\![a_2]\!]\sigma\} \tag{D3}$$

$$\mathcal{A}[\![a_1 \texttt{-} a_2]\!] = \{(\sigma, n_1 - n_2) \mid n_1 = \mathcal{A}[\![a_1]\!]\sigma,\ n_2 = \mathcal{A}[\![a_2]\!]\sigma\} \tag{D4}$$

$$\mathcal{A}[\![a_1 \texttt{*} a_2]\!] = \{(\sigma, n_1 n_2) \mid n_1 = \mathcal{A}[\![a_1]\!]\sigma,\ n_2 = \mathcal{A}[\![a_2]\!]\sigma\} \tag{D5}$$

Boolean Expressions

$$\mathcal{B}[\![\texttt{true}]\!] = \{(\sigma, T) \mid \sigma \in \Sigma\} \tag{D6}$$

$$\mathcal{B}[\![\texttt{false}]\!] = \{(\sigma, F) \mid \sigma \in \Sigma\} \tag{D7}$$

$$\begin{aligned}
\mathcal{B}[\![a_1 \texttt{<=} a_2]\!] = {} & \{(\sigma, T) \mid \mathcal{A}[\![a_1]\!]\sigma \leq \mathcal{A}[\![a_2]\!]\sigma\} \cup \\
& \{(\sigma, F) \mid \mathcal{A}[\![a_1]\!]\sigma > \mathcal{A}[\![a_2]\!]\sigma\}
\end{aligned} \tag{D8}$$

$$\begin{aligned}
\mathcal{B}[\![b_1 \texttt{\&\&} b_2]\!] = {} & \{(\sigma, T) \mid \mathcal{B}[\![b_1]\!]\sigma = T,\ \mathcal{B}[\![b_2]\!]\sigma = T\} \cup \\
& \{(\sigma, F) \mid \mathcal{B}[\![b_1]\!]\sigma = F\} \cup \\
& \{(\sigma, F) \mid \mathcal{B}[\![b_2]\!]\sigma = F\}
\end{aligned} \tag{D9}$$

$$\begin{aligned}
\mathcal{B}[\![b_1 \texttt{||} b_2]\!] = {} & \{(\sigma, T) \mid \mathcal{B}[\![b_1]\!]\sigma = T\} \cup \\
& \{(\sigma, T) \mid \mathcal{B}[\![b_2]\!]\sigma = T\} \cup \\
& \{(\sigma, F) \mid \mathcal{B}[\![b_1]\!]\sigma = F,\ \mathcal{B}[\![b_2]\!]\sigma = F\}
\end{aligned} \tag{D10}$$

$$\begin{aligned}
\mathcal{B}[\![\texttt{!}b]\!] = {} & \{(\sigma, T) \mid \mathcal{B}[\![b]\!]\sigma = F\} \cup \\
& \{(\sigma, F) \mid \mathcal{B}[\![b]\!]\sigma = T\}
\end{aligned} \tag{D11}$$

Commands

$$\mathcal{C}[\![\texttt{skip}]\!] = \{(\sigma, \sigma) \mid \sigma \in \Sigma\} \tag{D12}$$

$$\mathcal{C}[\![v := a]\!] = \{(\sigma, \sigma[v \mapsto n]) \mid n = \mathcal{A}[\![a]\!]\sigma\} \tag{D13}$$

$$\mathcal{C}[\![c_1; c_2]\!] = \{(\sigma, \mathcal{C}[\![c_2]\!](\mathcal{C}[\![c_1]\!]\sigma)) \mid \sigma \in \Sigma\} = \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!] \tag{D14}$$

$$\begin{aligned}
\mathcal{C}[\![\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2]\!] = {} & \{(\sigma, \mathcal{C}[\![c_1]\!]\sigma) \mid \mathcal{B}[\![b]\!]\sigma = T\} \cup \\
& \{(\sigma, \mathcal{C}[\![c_2]\!]\sigma) \mid \mathcal{B}[\![b]\!]\sigma = F\}
\end{aligned} \tag{D15}$$

$$\mathcal{C}[\![\texttt{while}\ b\ \texttt{do}\ c]\!] = \bigcup_{i \geq 0} \Gamma^i(\bot_{\Sigma \rightharpoonup \Sigma}) = \mathit{fix}(\Gamma) \tag{D16}$$

$$\begin{aligned}
\text{where} \quad \Gamma(g) = {} & \{(\sigma, (g \circ \mathcal{C}[\![c]\!])(\sigma)) \mid \mathcal{B}[\![b]\!]\sigma = T\} \cup \\
& \{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!]\sigma = F\}
\end{aligned}$$