Arithmetic Expression Summary

\[(n, \sigma) \mapsto \text{eval}(n)\]

\[(x, \sigma) \mapsto \sigma(x)\]

\[(a_0, \sigma) \mapsto k_0 \quad (a_1, \sigma) \mapsto k_1\]

\[(a_0 + a_1, \sigma) \mapsto k_0 + k_1\]

\[(a_0 - a_1, \sigma) \mapsto k_0 - k_1\]

\[(a_0 \times a_1, \sigma) \mapsto k_0 \times k_1\]

\[(a, \sigma) \mapsto k\]

\[((a), \sigma) \mapsto k\]

with \(k, k_0, k_1 \in \mathbb{I}, n, x, a, a_0, a_1 \in \mathbf{Aexp}, \) and \(\sigma \in \Sigma.\)
Our notion of semantic value for expressions leads to a natural equivalence relation between arithmetic expressions:

\[ a_0 \sim a_1 \text{ iff } \forall \sigma \in \Sigma, \exists n \in \mathbb{I}. (a_0, \sigma) \mapsto n \land (a_1, \sigma) \mapsto n, \]

where \( a_0, a_1 \in \text{Aexp}. \)

*Two expressions are equivalent if and only if they evaluate to the same semantic value in all possible states.*

(You should convince yourself that this is indeed an equivalence relation, i.e., check that the relation \( \sim \) is reflexive, symmetric, and transitive.)
**Problem:** Let $a_0 = 2 \times 3$ and $a_1 = 3 + 3$, with $a_0, a_1 \in A_{exp}$. Show that $a_0 \sim a_1$. 
Proof: We need to show that \((2 \times 3, \sigma) \mapsto k\) and \((3 + 3, \sigma) \mapsto k\) for all states \(\sigma \in \Sigma\) and some \(k \in \mathbb{I}\).

Let \(\sigma' \in \Sigma\) be any state, then

\[
\begin{align*}
(2, \sigma') & \mapsto 2 \\
(3, \sigma') & \mapsto 3 \\
(2 \times 3, \sigma') & \mapsto 6
\end{align*}
\]

and

\[
\begin{align*}
(3, \sigma') & \mapsto 3 \\
(3, \sigma') & \mapsto 3 \\
(3 + 3, \sigma') & \mapsto 6
\end{align*}
\]

which shows that regardless of the state, the two expressions will always produce the same semantics value, namely the integer 6. This concludes the proof. \(\square\)
**Problem:** Show that the $+$ operator is commutative.
Proof: We need to show that $a_0 + a_1 \sim a_1 + a_0$ for all $a_0, a_1 \in \mathbf{Aexp}$. We show this by demonstrating that

$$(a_0 + a_1, \sigma) \mapsto n \land (a_1 + a_0, \sigma) \mapsto n$$

for all $\sigma \in \Sigma$ and $n \in I$.

Assume that

$$(a_0, \sigma') \mapsto k_0$$

and

$$(a_1, \sigma') \mapsto k_1$$

for some $\sigma' \in \Sigma$ and $k_0, k_1 \in I$. 


Then we can construct the derivations

\[
\frac{(a_0, \sigma') \mapsto k_0 \quad (a_1, \sigma') \mapsto k_1}{(a_0 + a_1, \sigma') \mapsto k_0 + k_1}
\]

and

\[
\frac{(a_1, \sigma') \mapsto k_1 \quad (a_0, \sigma') \mapsto k_0}{(a_1 + a_0, \sigma') \mapsto k_1 + k_0 = k_0 + k_1}
\]

This proves the commutativity of $\mathbf{+}$. $\square$

**Observation:** Commutativity of the syntactic $\mathbf{+}$ operator is provided by the commutativity of the $\mathbf{+}$ operator over the set of integers.
Boolean Expressions

Recall our production for boolean expressions:

\[ B ::= \text{true} | \text{false} | A = A | A \leq A | \neg B | B \&\& B | B \| B | (B) \]

To compute the semantic value of boolean expressions we define an evaluation function ‘\( \mapsto \)’,

\[ \mapsto : \text{Bexp} \times \Sigma \rightarrow \mathbb{B}, \]

and write

\[ (be, \sigma) \mapsto t, \]

with \( be \in \text{Bexp} \), \( \sigma \in \Sigma \), and \( t \in \mathbb{B} \).

As in the case of the arithmetic expressions we introduce an \( \text{eval} \) function in order to map the syntactic representations of boolean constants in \( T \) into the semantic concepts of the constant in \( \mathbb{B} \),

\[ \text{eval} : \ T \rightarrow \mathbb{B} \]

---

1 What does the inductive definition of \( \text{Bexp} \) look like?
Boolean Expressions

\[(true, \sigma) \mapsto eval(true)\]

\[(false, \sigma) \mapsto eval(false)\]

\[(a_0, \sigma) \mapsto n \quad (a_1, \sigma) \mapsto m\]
\[\quad (a_0 = a_1, \sigma) \mapsto true\]
if \(n\) and \(m\) are equal

\[(a_0, \sigma) \mapsto n \quad (a_1, \sigma) \mapsto m\]
\[\quad (a_0 = a_1, \sigma) \mapsto false\]
if \(n\) and \(m\) are not equal

\[(a_0, \sigma) \mapsto n \quad (a_1, \sigma) \mapsto m\]
\[\quad (a_0 \leq a_1, \sigma) \mapsto true\]
if \(n\) is less than or equal to \(m\)

\[(a_0, \sigma) \mapsto n \quad (a_1, \sigma) \mapsto m\]
\[\quad (a_0 \leq a_1, \sigma) \mapsto false\]
if \(n\) is not less than or equal to \(m\)

with \(true, false \in T\), \(a_0, a_1 \in A_{\text{exp}}\), \(\sigma \in \Sigma\), and \(m, n \in \mathbb{I}\).
Boolean Expressions

\[
\begin{align*}
(b, \sigma) & \mapsto \text{true} & (b, \sigma) & \mapsto \text{false} \\
(!b, \sigma) & \mapsto \text{false} & (!b, \sigma) & \mapsto \text{true}
\end{align*}
\]

\[
\begin{align*}
(b_0, \sigma) & \mapsto t_0 & (b_1, \sigma) & \mapsto t_1 \\
(b_0 \& \& b_1, \sigma) & \mapsto t
\end{align*}
\]

where \( t \) is \text{true} if \( t_0 = \text{true} \) and \( t_1 = \text{true} \), and \text{false} otherwise.

\[
\begin{align*}
(b_0, \sigma) & \mapsto t_0 & (b_1, \sigma) & \mapsto t_1 \\
(b_0 \| b_1, \sigma) & \mapsto t
\end{align*}
\]

where \( t \) is \text{true} if \( t_0 = \text{true} \) or \( t_1 = \text{true} \), and \text{false} otherwise.

Here \( b, b_0, b_1 \in \text{Bexp}, t_0, t_1, t \in \mathbb{B}, \) and \( \sigma \in \Sigma \).
As in the case of $\textbf{Aexp}$, our notion of semantic value for expressions leads to an equivalence relation between boolean expressions:

$$b_0 \sim b_1 \text{ iff } \forall \sigma \in \Sigma, \exists t \in \mathbb{B}. \ (b_0, \sigma) \mapsto t \land (b_1, \sigma) \mapsto t,$$

where $b_0, b_1 \in \textbf{Bexp}$.

One way to look at this is that boolean expressions behave analogous to arithmetic expression except that the base has changed.
Recall our grammar production for commands:\(^2\):

\[
C ::= \text{skip} \mid V ::= A \mid C \mid \text{if } B \text{ then } C \text{ else } C \text{ end} \mid \text{while } B \text{ do } C \text{ end}
\]

In order to design a semantics for commands we have to answer the following questions:

1. What is the semantic domain for commands?
2. What does the evaluation function look like?

\(^2\)Inductive definition of the syntactic domain \textbf{Com}?
In our simple imperative language commands modify the state of the computation, that is, **commands map one state into another**. Therefore we define our evaluation function ‘$\mapsto$’ as,

$$\mapsto : \mathbf{Com} \times \Sigma \rightarrow \Sigma$$

and we write, given a command $c \in \mathbf{Com}$ and some state $\sigma \in \Sigma$,

$$(c, \sigma) \mapsto \sigma'$$

where $\sigma' \in \Sigma$ is the state after command $c$ has *fully executed*. 
Before we can give the full natural semantics for commands we need some more machinery. Consider, 

\[(x := 5, \sigma) \mapsto \sigma'\]

where $x \in \text{Loc}$, $5 \in I$, and $\sigma, \sigma' \in \Sigma$.

Here, $\sigma'$ is the state $\sigma$ updated to have the value 5 in location $x$. We write,

$\sigma' = \sigma[5/x]$. 
More formally, let $\sigma \in \Sigma$, $m \in \mathbb{I}$, and $x, y \in \textbf{Loc}$. We write $\sigma[m/x]$ for the state obtained from $\sigma$ by replacing the contents in $x$ with $m$. We can define this functionally,

$$
\sigma[m/x](y) = \begin{cases} 
m & \text{if } y = x \\
\sigma(y) & \text{if } y \neq x
\end{cases}
$$

$\Rightarrow$ States are “lookup tables” for values associated with locations.

Note that $\sigma[m/x] : \textbf{Loc} \to \mathbb{I}$ is still considered a function from locations into the integers.
Exercises: Let $\sigma' = \sigma_0[3/q]$ with $3 \in \mathbb{I}$ and $q \in \text{Loc}$,

- Compute the value of $\sigma'(q)$.
- Compute the value of $\sigma'(k)$ with $k \in \text{Loc}$ and $k \neq q$. 
Assume that all metavariables range over their appropriate domains and $\sigma, \sigma', \text{ and } \sigma'' \in \Sigma$.

\[
\begin{align*}
    (\text{skip}, \sigma) \mapsto & \quad \sigma \\
    (a, \sigma) \mapsto & \quad m \\
    (x := a, \sigma) \mapsto & \quad \sigma[m/x] \\
    (b, \sigma) \mapsto & \quad \text{true} \quad (c_0, \sigma) \mapsto \sigma' \\
    (\text{if } b \text{ then } c_0 \text{ else } c_1 \text{ end}, \sigma) \mapsto & \quad \sigma' \\
    (b, \sigma) \mapsto & \quad \text{false} \quad (c_1, \sigma) \mapsto \sigma' \\
    (\text{if } b \text{ then } c_0 \text{ else } c_1 \text{ end}, \sigma) \mapsto & \quad \sigma'
\end{align*}
\]
\[
\begin{align*}
(c_0, \sigma) & \implies \sigma'' \\
(c_1, \sigma'') & \implies \sigma' \\
(c_0; c_1, \sigma) & \implies \sigma' \\
(b, \sigma) & \implies \text{false} \\
(\text{while } b \text{ do } c \text{ end}, \sigma) & \implies \sigma \\
(b, \sigma) & \implies \text{true} \\
(c, \sigma) & \implies \sigma'' \\
(\text{while } b \text{ do } c \text{ end}, \sigma'') & \implies \sigma' \\
(\text{while } b \text{ do } c \text{ end}, \sigma) & \implies \sigma'
\end{align*}
\]