The last page of this document contains a reference description of the syntax and semantics of two different arithmetic systems, one with only natural numbers, and one with booleans. To carefully distinguish between these systems, we will call them System 1 and System 2, respectively. Problem 9 asks you design System 3.

Problem 6. Consider the following statement:

For every expression e, there exists a value v such that $e \longrightarrow^* v$.

(a) Consider System 1. Explain why the theorem we proved in class, that $e \longrightarrow^* eval(e)$, implies the statement above.

Since $eval(e)$ yields a value in meta-language, and $e \longrightarrow^* eval(e)$, subsequently, we can conclude that for every expression e, there exists a value v such that $e \longrightarrow^* v$.

(b) Now consider System 2. Explain why the statement above is not true in System 2.

In System 2, the expression can be ill-typed, which will result in a stuck. Therefore the statement might not be true for System 2.

(c) Still considering System 2, tweak the statement above to restrict it to well-typed expressions. State your tweaked version in full, using wording as similar to the above statement as possible.

For every well-typed expression e, there exists a value v such that $e \longrightarrow^* v$.

(d) Prove (by induction on a thing of your choice) this modified statement for System 2.

Since e is a well-typed expression, $\vdash e : \tau$. Thus we are going to prove if $\vdash e : \tau$, then there exists a value v such that $e \longrightarrow^* v$. Induction on this typing relation.

Proof.

- Base case.

$$\frac{}{\vdash e : \mathrm{int}} \qquad\qquad \frac{}{\vdash e : \mathrm{bool}}$$

By the definition of $\longrightarrow^*$, since e itself is a value, $e \longrightarrow^* e$. Thus if e in this case is well-typed, then exists the value e such that $e \longrightarrow^* e$.

- Case

$$\frac{\vdash e_1 : \mathrm{int} \qquad \vdash e_2 : \mathrm{int}}{\vdash e_1 + e_2 : \mathrm{int}}$$

With induction hypothesis that if $\vdash e_1 : \mathrm{int}$, then exists v_1 such that $e_1 \longrightarrow^* v_1$; if $\vdash e_2 : \mathrm{int}$, then exists v_2 such that $e_2 \longrightarrow^* v_2$.

According to our assumption, $\vdash e_1 : \mathrm{int}$ and $\vdash e_2 : \mathrm{int}$. Then according to the induction hypothesis, exists v_1 such that $e_1 \longrightarrow^* v_1$; exists v_2 such that $e_2 \longrightarrow^* v_2$. Subsequently, $e_1 + e_2 \longrightarrow^* v_1 + e_2$; $v_1 + e_2 \longrightarrow^* v_2$. Since v_1 and v_2 are int values, according to the definition of $\longrightarrow$, $v_1 + v_2 \longrightarrow v_1 + v_2$. Therefore, if $e_1 + e_2$ is well-typed, then exists $v_1 + v_2$ such that $e_1 + e_2 \longrightarrow^* v_1 + v_2$.

- Case

$$\frac{\vdash e_1 : \mathrm{bool} \qquad \vdash e_2 : \mathrm{bool}}{\vdash e_1 \wedge e_2 : \mathrm{bool}}$$

With induction hypothesis that if $\vdash e_1 : \mathrm{bool}$, then exists b_1 such that $e_1 \longrightarrow^* b_1$; if $\vdash e_2 : \mathrm{bool}$, then exists b_2 such that $e_2 \longrightarrow^* b_2$.

According to our assumption, $\vdash e_1 : \mathrm{bool}$, $\vdash e_2 : \mathrm{bool}$, and the induction hypothesis, exists a value b_1 such that $e_1 \longrightarrow^* b_1$; exists a value b_2 such that $e_2 \longrightarrow^* b_2$. Subsequently,

$e_1 \wedge e_2 \to^* b_1 \wedge e_2$, and then $b_1 \wedge e_2 \to^* b_1 \wedge b_2$. Since b_1 and b_2 are bool values, according to the definition of $\to$, $b_1 \wedge b_2 \to b_2 \wedge b_2$. Thus if $e_1 \wedge e_2$ is well-typed, exists a value $b_1 \wedge b_2$ such that $e_1 \wedge e_2 \to^* b_1 \wedge b_2$.

□

(e) Explain informally why your modified statement from part (c) is *not* equivalent to the type safety theorem we proved in class on Friday (April 3rd) about System 2. (Hint: Start by stating type safety using wording as similar as possible to your statement. Could there be a small-step semantics for some programming language that satisfies type safety but not your statement? What about vice versa?)

It is not equivalent to the type safety theorem because the modified statement from part (c) guarantees termination whereas type safty theorem, applied to some programming languages (e.g. STLC with fixpoint combinator), does not guarantee termination.

(f) Consider your proof from part (d). List all the (relevant) things you *could* have tried to induct on. For each one, say briefly whether you think it would have worked to induct on that thing (it's ok to be wrong; just say what you think). Finally, say why you chose to induct on what you did.

$\vdash e : \tau$, e. **I chose the typing rules because it is related more closely to evaluation, so it is more likely to be a proof without extra efforts.**

Problem 7. This problem about the notion of subexpressions, and considers only System 2.

(a) Use long horizontal lines to define a relation $e_1 \leqslant e_2$, which should encode the idea that "e_1 is a subexpression of e_2".

Hint: You should have 5 rules. One should be an axiom for reflexivity (everything is a subexpression of itself). Then, there should be two recursive rules for each of the binary operations in the syntax of System 2, one which says "subexpressions of the left child of an AST node are also subexpressions of the node itself", and similarly for "right child".

$$\frac{}{e \leqslant e}$$

$$\frac{e_1' \leqslant e_1}{e_1' \leqslant e_1 + e_2} \qquad \frac{e_2' \leqslant e_2}{e_2' \leqslant e_1 + e_2}$$

$$\frac{e_1' \leqslant e_1}{e_1' \leqslant e_1 \wedge e_2} \qquad \frac{e_2' \leqslant e_2}{e_2' \leqslant e_1 \wedge e_2}$$

(b) Prove (by induction on a thing of your choice) the following statement:

For all expressions e_1 and e_2, and for all types τ, if $e_1 \leqslant e_2$ and $\vdash e_2 : \tau$, then $\vdash e_1 : \tau$.

Proof. By induction on the typing rule $\vdash e_2 : \tau$.

i. Base Case. $\frac{}{\vdash e_2 : \text{int}}$ and $\frac{}{\vdash e_2 : \text{bool}}$. In this case, the only possible way to get $e_1 \leqslant e_2$ is $\frac{}{e \leqslant e}$.

Then we know that e_1 and e_2 is the same expression, hence have the same type τ. Therefore, $\vdash e_1 : \tau$.

ii. Case. $\frac{\vdash e_l : \text{int} \qquad \vdash e_r : \text{int}}{\vdash e_l + e_r : \text{int}}$. In this case, the assumption is $e_1 \leqslant e_l + e_r$. By doing case analysis on this subexpression relation:

1. $\dfrac{e_1 \leqslant e_l}{e_1 \leqslant e_l + e_r}$.
   Since $e_1 \leqslant e_l$ and $\vdash e_l : \mathsf{int}$, according to the induction hypothesis, $\vdash e_1 : \mathsf{int}$.
2. $\dfrac{e_1 \leqslant e_r}{e_1 \leqslant e_l + e_r}$.
   Similarly, according to the induction hypothesis, since $e_1 \leqslant e_r$ and $\vdash e_r : \mathsf{int}$, we can conclude that $\vdash e_1 : \mathsf{int}$.

3.& 4. The rest cases are not admissible since $e_1 \leqslant e_l + e_r$ does not follow the construction.

iii. Case. $\dfrac{\vdash e_l : \mathsf{bool} \qquad \vdash e_r : \mathsf{bool}}{\vdash e_l \wedge e_r : \mathsf{int}}$. The proof is similar to that of Case ii.

Therefore, the following conclusion is that for all expressions e_1 and e_2, and for all types τ, if $e_1 \leqslant e_2$ and $\vdash e_2 : \tau$, then $\vdash e_1 : \tau$ □

(c) Consider your proof of part (b). List all the (relevant) things you *could* have tried to induct on. For each one, say briefly whether you think it would have worked to induct on that thing (it's ok to be wrong; just say what you think). Finally, say why you chose to induct on what you did.

I could have done induction on e_1, e_2, the typing rule $\vdash e_2 : \tau$ and the subexpression rule $e_1 \leqslant e_2$. I chose the typing rule, since it does not require proofs of other properties of the subexpression relations. I tried to prove the theorem by induction on the subexpression relation, but it turns out that I need to prove things like if $\vdash e_1 + e_2 : \mathsf{int}$, the $\vdash e_1 : \mathsf{int}$ and $\vdash e_2 : \mathsf{int}$, which requires some extra work to do.

(d) Consider this modified version of the statement from part (b), where we have switched e_1 and e_2 in a few key places.

For all expressions e_1 and e_2, and for all types τ, if $e_1 \leqslant e_2$ and $\vdash e_1 : \tau$, then $\vdash e_2 : \tau$.

Give an example e_1, e_2, and τ that shows this statement is false.

Let τ be int.
Let e_1 be $1 + 2$.
Let e_2 be $1 + 2 \wedge \top$

Problem 8. Consider the following statement of "type completeness" for System 2.

For every expression e, value v, and type τ, if $e \longrightarrow^* v$ and $\vdash v : \tau$, then $\vdash e : \tau$.

(a) Explain informally what this statement means.

If an expression e can take step(s) to a value v, then the type of e must be the same as the type of v

(b) List all the (relevant) things you *could* try to induct on in order to prove this statement. For each one, say briefly whether you think it would work to induct on that thing (it's ok to be wrong; just say what you think).

I could try to induct on e, $e \longrightarrow^* v$. For e, it is related to the structure of the expression so I might be able to get some evidence while steping through the evaluation. For $e \longrightarrow^* v$, it is directly related to the semantics, so maybe there are some straightfoward evidences while doing induction.

(c) Prove this statement (by induction on a thing of your choice). (Hint: You will need at least one lemma, which itself should be proved by induction. State your lemma as clearly as possible. Also make it clear what you are inducting on in the proof of your lemma.)

Lemma 1. For all expressions e, e' and all types τ. If $e \longrightarrow e'$ and $\vdash e' : \tau$, then $\vdash e : \tau$.

Proof. First split cases based on τ.

Case: when τ is $\mathtt{int}$. By induction on $\vdash e' : \mathtt{int}$.

- $\dfrac{}{\vdash e' : \mathtt{int}}$. Since e' in this case is already a value, so the only rule for $e \longrightarrow e'$ is $\dfrac{}{n_1 + n_2 \longrightarrow n_1 + n_2}$, hence $e = n_1 + n_2$. In this case $\vdash n_1 : \mathtt{int}$ and $\vdash n_2 : \mathtt{int}$, so by applying $\dfrac{\vdash n_1 : \mathtt{int} \qquad \vdash n_2 : \mathtt{int}}{\vdash n_1 + n_2 : \mathtt{int}}$, we get $\vdash e : \mathtt{int}$

- $\dfrac{\vdash e_1 : \mathtt{int} \qquad \vdash e_2 : \mathtt{int}}{\vdash e_1 + e_2 : \mathtt{int}}$. By case analysis on $e \longrightarrow e_1 + e_2$:
  i. $n_1 + n_2 = n_1 + n_2$. Since the right hand side, $e_1 + e_2$ is not a value, which contradicts to our assumption.
  ii. $\dfrac{e_\alpha \longrightarrow e_1}{e_\alpha + e_2 \longrightarrow e_1 + e_2}$. According to assumptions, $e_\alpha \longrightarrow e_1$ and $\vdash e_1 : \mathtt{int}$, and the induction hypothesis, we get $\vdash e_\alpha : \mathtt{int}$. Since $\vdash e_2 : \mathtt{int}$ according to assumption, applying the rule $\dfrac{\vdash e_\alpha : \mathtt{int} \qquad \vdash e_2 : \mathtt{int}}{\vdash e_\alpha + e_2 : \mathtt{int}}$. Thus $\vdash e : \mathtt{int}$.
  iii. $\dfrac{e_\beta \longrightarrow e_2}{n_1 + e_\beta \longrightarrow n_1 + e_2}$. Similarly by induction hypothesis and assumption that $e_\beta \longrightarrow e_2$ and $\vdash e_2 : \mathtt{int}$, we can get $\vdash e_\beta : \mathtt{int}$. Thus by applying $\dfrac{\dfrac{}{\vdash n_1 : \mathtt{int}} \qquad \dfrac{}{\vdash e_\beta : \mathtt{int}}}{\vdash n_1 + e_\beta : \mathtt{int}}$, subsequently $\vdash e : \mathtt{int}$.

Case: when τ is $\mathtt{bool}$. The proof is similar to that of the case when τ is $\mathtt{int}$. □

Proof. By induction on $e \longrightarrow^* v$.

- $\dfrac{}{e \longrightarrow^* e}$. In this case e is v. Since $\vdash v : \tau$, $\vdash e : \tau$.
- $\dfrac{e \longrightarrow e' \qquad e' \longrightarrow^* v}{e \longrightarrow^* v}$. According to induction hypothesis, and assumptions that $e' \longrightarrow^* v$ and $\vdash v : \tau$, we can get $\vdash e' : \tau$. According to the assumption that $e \longrightarrow e'$ and $e' : \tau$ and *Lemma 1* proved above, $e : \tau$ follows.

□

Problem 9. This problem asks you to modify System 2 to change the semantics of $\wedge$ slightly.

(a) We mentioned briefly in class that the semantics of $+$ and $\wedge$ enforce a left-to-right evaluation order. In most programming languages, && has "short-circuiting" semantics, where the right subexpression is evaluated only if the left subexpression evaluates to true. Explain informally why the rules of System 2 do *not* allow short-circuiting evaluation.

Because System 2 enforces that the reduction from an expresion to a value in meta-language can be done only when both left hand side and right hand side are bool **values, right hand side will always be evaluated before the expression reduces to a value.**

(b) Design a modified version of System 2 (call it System 3!) that changes the semantics of $\wedge$ to be short-circuiting. Clearly state which rules relating to $\wedge$ you are deleting from the system (they have been given little names on the last page), and which rules you are adding.

Delete the rule And_2**. Add following two rules:**

$$\frac{}{\bot \wedge e \to \bot}\ \textbf{AndShortCircuiting} \qquad \frac{e \to e'}{\top \wedge e \to \top \wedge e'}\ \textbf{AndCont}$$

(c) Give an example of a program that exercises the short-circuiting semantics of $\wedge$.
Let the program be $\top \wedge (\bot \wedge (\top \wedge \top))$
Then the evaluation becomes:

$$\dfrac{\dfrac{\dfrac{}{\bot \wedge (\top \wedge \top) \to \bot}}{\top \wedge (\bot \wedge (\top \wedge \top)) \to \top \wedge \bot} \qquad \dfrac{\dfrac{}{\top \wedge \bot \to \bot} \quad \dfrac{}{\bot \to^* \bot}}{\top \wedge \bot \to^* \bot}}{\top \wedge (\bot \wedge (\top \wedge \top)) \to^* \bot}$$

(d) Consider the informal statement:

If a program evaluates to a value in System 2, then it evaluates to the same value in System 3.

State this claim more formally by introducing (meta-)variables and using $\to^*$. (There is more than one way to state it formally; choose the way that seems simplest and clearest to you. Whatever you do, be sure to clearly distinguish which uses of $\to$ refer to System 2 as opposed to System 3, perhaps by using $\to_2$ and $\to_3$.)

For all expressions e**. If** $e \to_2^* v$**, then** $e \to_3^* v$ **for some value** v

(e) Prove (by induction on a thing of your choice) your formalized claim.

Lemma 9-0.1: $\forall e_1, e_2.\ \dfrac{e_1 \to_3^* e_1'}{e_1 \wedge e_2 \to_3^* e_1' \wedge e_2}$ for some expression e_1'.

Proof. By induction on the steping rule $e_1 \to_3^* e_1'$

i. $e_1 \to_3^* e_1$. e_1' is e_1. Thus $e_1 \wedge e_2 \to_3^* e_1' \wedge e_2$ holds by the reflexivity rule.

ii. $\dfrac{e_1 \to_3 e' \qquad e' \to_3^* e_1'}{e_1 \wedge e_2 \to_3^* e_1' \wedge e_2}$. By using the steping rule and the induction hypothesis, we can conclude

$$\dfrac{\dfrac{e_1 \to_3 e'}{e_1 \wedge e_2 \to_3 e' \wedge e_2} \qquad \dfrac{e' \to_3^* e_1'}{e' \wedge e_2 \to_3^* e_1' \wedge e_2}}{e_1 \wedge e_2 \to_3^* e_1' \wedge e_2}$$

□

Lemma 9-1: $\forall e, \tau.\ \dfrac{\vdash e : \tau}{\exists v.\ e \rightarrow_2^* v \wedge e \rightarrow_3^* v}$.

Proof. By case analysis on τ.

i. τ is `int`. For all rules that related to `int`, System 2 and System 3 are identical. Thus the claim holds by the type safety theorem.

ii. τ is `bool`. By induction on the typing rule $\vdash e : \text{bool}$.

- $\dfrac{}{e : \text{bool}}$. In this case, v is e; therefore $e \rightarrow_2^* e$ and $e \rightarrow_3^* e$.
- $\dfrac{\vdash e_1 : \text{bool} \qquad \vdash e_2 : \text{bool}}{\vdash e_1 \wedge e_2 : \text{bool}}$. By the induction hypothesis,

$$\exists v_1.\ e_1 \rightarrow_2^* v_1 \wedge e_1 \rightarrow_3^* v_1$$

$$\exists v_2.\ e_2 \rightarrow_2^* v_2 \wedge e_2 \rightarrow_3^* v_2$$

Therefore, we can rewrite the expression to $\dfrac{\vdash v_1 : \text{bool} \qquad \vdash e_2 : \text{bool}}{\vdash v_1 \wedge e_2 : \text{bool}}$ by *Lemma 9-0.1*.

By case analysis on v_1:

A. v_1 is $\top$. The expression becomes $\top \wedge e_2$. According to the induction hypothesis, the expression can be rewritten to $\top \wedge v_2$. Since the evaluation of base case is the same in System 2 and System 3, therefore the expression evaluates to the same value under System 2 and System 3.

B. v_1 is $\bot$. In this case, under System 3, the expression immediately evaluates to $\bot$. Under System 2, the expression can be rewritten to $\bot \wedge v_2$. In System 2, this evaluates to $\bot \wedge v_2$. According to the definition of $\wedge$ in meta-language, the expression evaluates to $\bot$. Thus exists the value $\bot$ such that the expression evaluates to this value in System 2 and System 3.

Therefore, if e is well-typed, there exists a value v such that $e \rightarrow_2^* v$ and $e \rightarrow_3^* v$.

□

Theorem. For all expressions e. If $e \rightarrow_2^* v$, then $e \rightarrow_3^* v$ for some value v

Proof. Since $e \rightarrow_2^* v$, according to the type completeness theorem, we only need to consider well-typed expressions. Therefore, assume e to be a well-typed expression, hence $\vdash e : \tau$ for some type τ. According to *Lemma 9-1*, we know that $e \rightarrow_2^* v$ and $e \rightarrow_3^* v$ for some value v. This conclusion implies this theorem, which is $e \rightarrow_2^* v \rightarrow e \rightarrow_3^* v$ for some value v. □

(f) List all the (relevant) things you *could* have tried to induct on in part (e). For each one, say briefly whether you think it would have worked to induct on that thing (it's ok to be wrong; just say what you think). Finally, say why you chose to induct on what you did.

I could induction on e, $e \rightarrow_2^* v$ or $\vdash e : \tau$. For e, it might be the case that I can apply the new rules directly by doing induction and case analysis. For the steping rules, I think it could be easier since I only modify one rule and I only need to show that under that rule, the result is the same in two systems. Since $e \rightarrow_2^* v$, from the type completeness theorem, we know e is a well-typed expression, thus I can also induction on the typing rules. I chose to induct on the typing rule to derive a lemma that is applicable for this theorem (and the one below), since it can lead to a stronger conclusion.

(g) Now consider the converse of the statement, in other words:

> If a program evaluates to a value in System 3, then it evaluates to the same value in System 2.

State this claim formally.

For all expressions e and all values v, if $e \rightarrow_3^* v$, then $e \rightarrow_2^* v$.

(h) Show that this claim is false by giving an example that evaluates differently in System 2 than it does in System 3. (Hint: Use short-circuiting to hide something bad from System 3!)

Consider the expression $\bot \wedge (1 + 2)$.
The evaluation in System 2 cannot be proceeded at this point, since it does not match any rule.
The evaluation in System 3 will be

$$\frac{}{\bot \wedge (1 + 2) \rightarrow_3^* \bot}$$

(i) Explain why your example does not contradict the claim from part (d).

According to the type completeness theorem proved in privious part, e is a well-typed expression. However, the example given in part (h) is not a well-typed expression, thus cannot apply to the statement in part (d).

(j) Explain how your example *also* answers the question "Does type completeness hold in System 3?".

Type completeness does not hold in System 3 since a ill-typed expression can evaluate to a well-typed value.

(k) Finally, consider restricting to well-typed programs. (System 2 and 3 have the same type system.)

> If a well-typed program evaluates to a value in System 3, then it evaluates to the same value in System 2.

Formalize and prove this claim (by induction on a thing of your choice).

Theorem. For all expressions e, if $\vdash e : \tau$ and $e \rightarrow_3^* v$, then $e \rightarrow_2^* v$ for some values v and types τ

Proof. Since $\vdash e : \tau$, according to *Lemma 9-1*, we know that $\exists v.\ e \rightarrow_2^* v \wedge e \rightarrow_3^* v$ for some value v. Thus, the conclusion of *Lemma 9-1* implies this theorem. □

(l) Explain where your proof from part (k) breaks down if you try to use it to prove the claim from part (g). (Don't just say "it breaks down because we need the program to be well typed". Point exactly to the step of the proof that is no longer true.)

System 1

Syntax and two styles of semantics for arithmetic with natural numbers and addition.

$$\begin{array}{rcl} e & ::= & n \mid e + e \\ v & ::= & n \\ n & \in & \mathbb{N} \end{array}$$

$$\begin{aligned} \mathrm{eval}(n) &= n \\ \mathrm{eval}(e_1 + e_2) &= \mathrm{eval}(e_1) + \mathrm{eval}(e_2) \end{aligned}$$

$\boxed{e \to e}$

$$\frac{}{n_1 + n_2 \to n_1 + n_2}$$

$$\frac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$$

$$\frac{e_2 \to e_2'}{n_1 + e_2 \to n_1 + e_2'}$$

System 2

Adding booleans to arithmetic, with small-step semantics and type system.

$$\begin{array}{rcl} e & ::= & \ldots \mid b \mid e \wedge e \\ v & ::= & \ldots \mid b \\ b & \in & \mathbb{B} \end{array}$$

$\boxed{e \to e}$

$$\ldots$$

$$\frac{}{b_1 \wedge b_2 \to b_1 \wedge b_2}\ \textsc{AndReduce}$$

$$\frac{e_1 \to e_1'}{e_1 \wedge e_2 \to e_1' \wedge e_2}\ \textsc{And}_1$$

$$\frac{e_2 \to e_2'}{b_1 \wedge e_2 \to b_1 \wedge e_2'}\ \textsc{And}_2$$

$$\tau \ ::= \ \mathsf{int} \mid \mathsf{bool}$$

$\boxed{\vdash e : \tau}$

$$\frac{}{\vdash n : \mathsf{int}} \qquad \frac{}{\vdash b : \mathsf{bool}}$$

$$\frac{\vdash e_1 : \mathsf{int} \quad \vdash e_2 : \mathsf{int}}{\vdash e_1 + e_2 : \mathsf{int}}$$

$$\frac{\vdash e_1 : \mathsf{bool} \quad \vdash e_2 : \mathsf{bool}}{\vdash e_1 \wedge e_2 : \mathsf{bool}}$$