Lecture 3a: Semantics & WHILE3ADDR

17-355/17-665/17-819: Program Analysis

Rohan Padhye

Jan 25, 2022

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Review: WHILE abstract syntax

S	statements
a	arithmetic expressions (AExp)
x, y	program variables (Vars)
n	number literals
b	boolean expressions (BExp)

$$
\begin{array}{lcl}
S & ::= & x := a \\
 & \mid & \text{skip} \\
 & \mid & S_1;\; S_2 \\
 & \mid & \text{if } b \text{ then } S_1 \text{ else } S_2 \\
 & \mid & \text{while } b \text{ do } S
\end{array}
\qquad
\begin{array}{lcl}
b & ::= & \text{true} \\
 & \mid & \text{false} \\
 & \mid & \text{not } b \\
 & \mid & b_1 \; op_b \; b_2 \\
 & \mid & a_1 \; op_r \; a_2
\end{array}
\qquad
\begin{array}{lcl}
a & ::= & x \\
 & \mid & n \\
 & \mid & a_1 \; op_a \; a_2
\end{array}
\qquad
\begin{array}{lcl}
op_b & ::= & \text{and} \mid \text{or} \\
op_r & ::= & < \;\mid\; \leqslant \;\mid\; = \\
 & \mid & > \;\mid\; \geqslant \\
op_a & ::= & + \mid - \mid * \mid /
\end{array}
$$

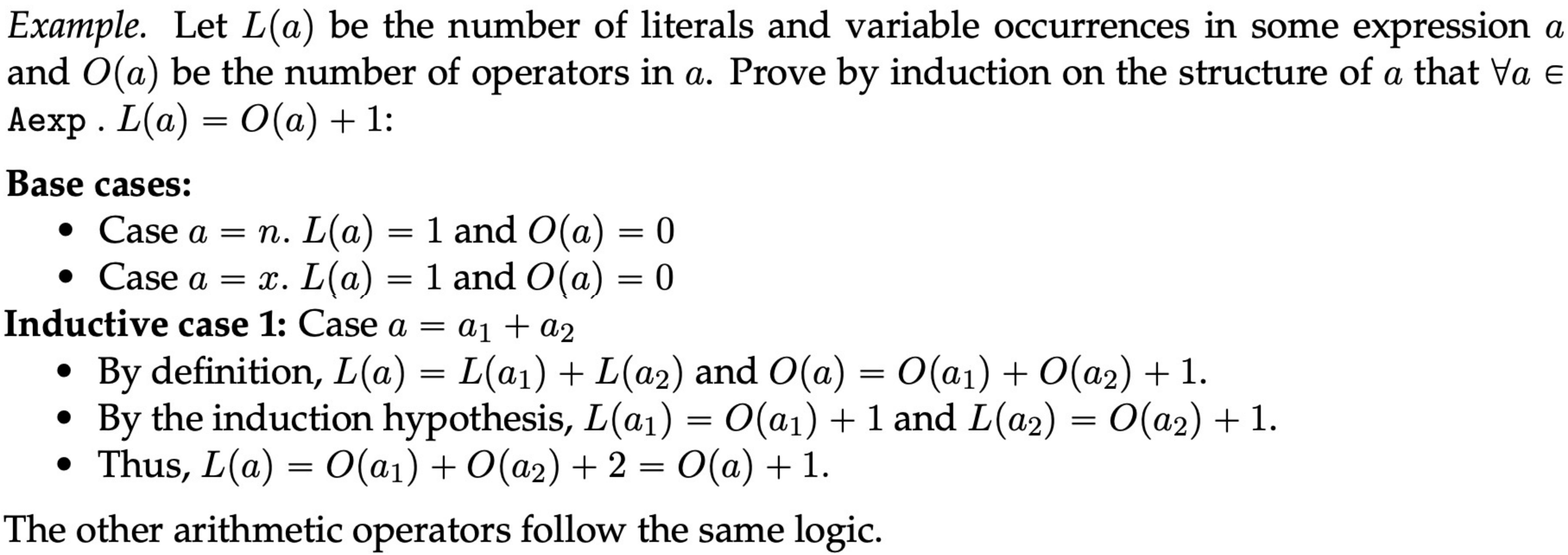
Review: Proofs by Structural Induction

$$
\begin{array}{rcl}
a & ::= & x \\
  & | & n \\
  & | & a_1 \; op_a \; a_2
\end{array}
\qquad
op_a \;\; ::= \;\; + \mid - \mid * \mid /
$$

- To prove $\forall a \in Aexp: P(a)$ by induction on structure of syntax
  - Base cases: show that $P(x)$ and $P(n)$ holds
  - Inductive cases: show that
    - $P(a_1) \wedge P(a_2) \Rightarrow P(a_1 + a_2)$
    - $P(a_1) \wedge P(a_2) \Rightarrow P(a_1 * \; a_2)$

    - $P(a_1) \wedge P(a_2) \Rightarrow P(a_1 / a_2)$

Review: Proofs by Structural Induction



Example. Let $L(a)$ be the number of literals and variable occurrences in some expression a and $O(a)$ be the number of operators in a. Prove by induction on the structure of a that $\forall a \in \texttt{Aexp} \,.\, L(a) = O(a) + 1$:

Base cases:

- Case $a = n$. $L(a) = 1$ and $O(a) = 0$
- Case $a = x$. $L(a) = 1$ and $O(a) = 0$

Inductive case 1: Case $a = a_1 + a_2$

- By definition, $L(a) = L(a_1) + L(a_2)$ and $O(a) = O(a_1) + O(a_2) + 1$.
- By the induction hypothesis, $L(a_1) = O(a_1) + 1$ and $L(a_2) = O(a_2) + 1$.
- Thus, $L(a) = O(a_1) + O(a_2) + 2 = O(a) + 1$.

The other arithmetic operators follow the same logic.

Review: Proofs by Structural Induction

- Prove that small-step and big-step semantics of expressions produce equivalent results.

$$\forall a \in \mathtt{\textbf{AExp}} \,.\, \langle E, a \rangle \rightarrow_a^* n \Leftrightarrow \langle E, a \rangle \Downarrow n$$

- Can be proved via structural induction over syntax. (Exercise)

Proofs by Structural Induction

- Prove that WHILE is *deterministic*. That is, if the program terminates, it evaluates to a unique value.

$$
\begin{array}{lll}
\forall a \in \texttt{Aexp} \,. & \forall E \,. \forall n, n' \in \mathbb{N} \,. & \langle E, a\rangle \Downarrow n \land \langle E, a\rangle \Downarrow n' \Rightarrow n = n' \\
\forall P \in \texttt{Bexp} \,. & \forall E \,. \forall b, b' \in \mathcal{B} \,. & \langle E, P\rangle \Downarrow b \land \langle E, P\rangle \Downarrow b' \Rightarrow b = b' \\
\forall S \,. & \forall E, E', E'' \,. & \langle E, S\rangle \Downarrow E' \land \langle E, S\rangle \Downarrow E'' \Rightarrow E' = E''
\end{array}
$$

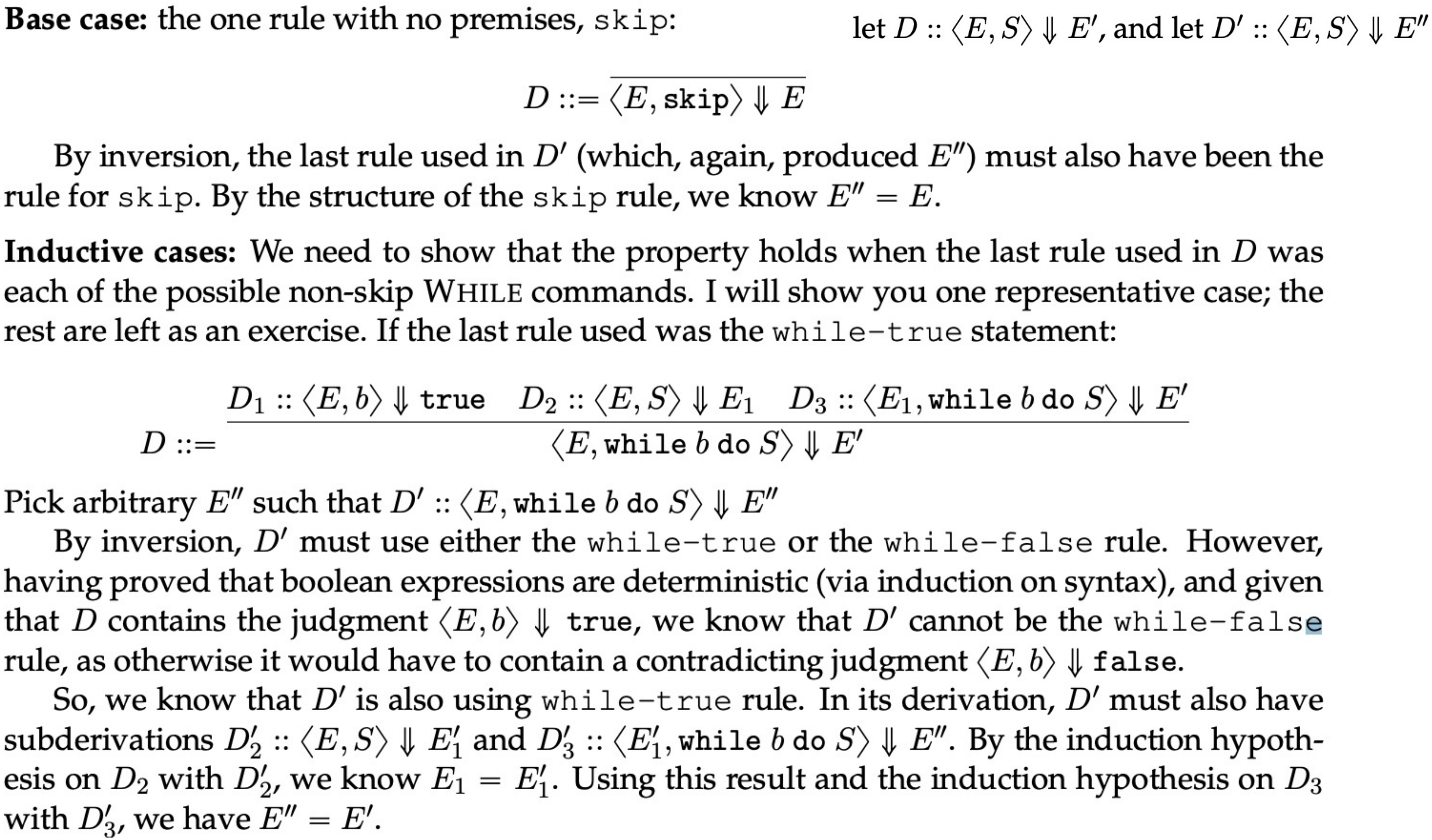
Rule for while is recursive; doesn't depend only on subexpressions

$$
\frac{\langle E, b\rangle \Downarrow \texttt{true} \quad \langle E, S; \texttt{while}\ b\ \texttt{do}\ S\rangle \Downarrow E'}{\langle E, \texttt{while}\ b\ \texttt{then}\ S\rangle \Downarrow E'}\ \textit{big-whiletrue}
$$

- Can prove for expressions via induction over syntax, but not for statements.
- But there's still a way.

To prove: $\forall S \,.\quad \forall E, E', E'' \,.\quad \langle E, S\rangle \Downarrow E' \wedge \langle E, S\rangle \Downarrow E'' \Rightarrow E' = E''$

Structural Induction over Derivations



Base case: the one rule with no premises, `skip`:

let $D :: \langle E, S\rangle \Downarrow E'$, and let $D' :: \langle E, S\rangle \Downarrow E''$

$$D ::= \overline{\langle E, \texttt{skip}\rangle \Downarrow E}$$

By inversion, the last rule used in D' (which, again, produced E'') must also have been the rule for `skip`. By the structure of the `skip` rule, we know $E'' = E$.

Inductive cases: We need to show that the property holds when the last rule used in D was each of the possible non-skip WHILE commands. I will show you one representative case; the rest are left as an exercise. If the last rule used was the `while-true` statement:

$$D ::= \frac{D_1 :: \langle E, b\rangle \Downarrow \texttt{true} \quad D_2 :: \langle E, S\rangle \Downarrow E_1 \quad D_3 :: \langle E_1, \texttt{while}\ b\ \texttt{do}\ S\rangle \Downarrow E'}{\langle E, \texttt{while}\ b\ \texttt{do}\ S\rangle \Downarrow E'}$$

Pick arbitrary E'' such that $D' :: \langle E, \texttt{while}\ b\ \texttt{do}\ S\rangle \Downarrow E''$

By inversion, D' must use either the `while-true` or the `while-false` rule. However, having proved that boolean expressions are deterministic (via induction on syntax), and given that D contains the judgment $\langle E, b\rangle \Downarrow \texttt{true}$, we know that D' cannot be the `while-false` rule, as otherwise it would have to contain a contradicting judgment $\langle E, b\rangle \Downarrow \texttt{false}$.

So, we know that D' is also using `while-true` rule. In its derivation, D' must also have subderivations $D_2' :: \langle E, S\rangle \Downarrow E_1'$ and $D_3' :: \langle E_1', \texttt{while}\ b\ \texttt{do}\ S\rangle \Downarrow E''$. By the induction hypothesis on D_2 with D_2', we know $E_1 = E_1'$. Using this result and the induction hypothesis on D_3 with D_3', we have $E'' = E'$.

Review: WHILE abstract syntax

S	statements
a	arithmetic expressions (AExp)
x, y	program variables (Vars)
n	number literals
b	boolean expressions (BExp)

$$
\begin{array}{rcl}
S & ::= & x := a \\
 & \mid & \text{skip} \\
 & \mid & S_1;\ S_2 \\
 & \mid & \text{if } b \text{ then } S_1 \text{ else } S_2 \\
 & \mid & \text{while } b \text{ do } S
\end{array}
\qquad
\begin{array}{rcl}
b & ::= & \text{true} \\
 & \mid & \text{false} \\
 & \mid & \text{not } b \\
 & \mid & b_1\ op_b\ b_2 \\
 & \mid & a_1\ op_r\ a_2
\end{array}
\qquad
\begin{array}{rcl}
a & ::= & x \\
 & \mid & n \\
 & \mid & a_1\ op_a\ a_2
\end{array}
\qquad
\begin{array}{rcl}
op_b & ::= & \text{and} \mid \text{or} \\
op_r & ::= & < \ \mid\ \leqslant \ \mid\ = \\
 & \mid & > \ \mid\ \geqslant \\
op_a & ::= & + \mid - \mid * \mid /
\end{array}
$$

WHILE syntax

- Abstract representation that corresponds well to concrete syntax
- Useful for recursive or inductive reasoning
- Sometimes challenging to track how data and control flows in program execution order
- 3-address-code is commonly used by compilers to represent imperative language code.
  - AST -> 3-address transformation is straightforward.

WHILE3ADDR

- w = x * y + z

- if b then S1 else S2

- 1: t = x * y
  2: w = t + z

- 1: if b then goto 4
  2: S2
  3: goto 5
  4: S1
  5: …

WHILE3ADDR:
An Intermediate Representation

- Simpler, more uniform than WHILE syntax
- Categories:
  - *I* ∈ **Instruction** instructions
  - *x*, *y* ∈ **Var** variables
  - *n* ∈ **Num** number literals
- Syntax:
  - I ::= x := n | x := y | x := y op z
    | goto n / if x op_r 0 goto n
  - op_a ::= + | – | * | / | …
  - op_r ::= < | ≤ | = | > | ≥ | ...
  - P ∈ **Num** → *I*

Exercise: Translate ***while b do S*** to WHILE3ADDR

- Categories:

○	*I* ∈ **Instruction**	instructions
○	*x*, *y* ∈ **Var**	variables
○	*n* ∈ **Num**	number literals

- Syntax:
  - I ::= x := n | x := y | x := y op z
    | goto n / if x $\texttt{op}_{r}$ 0 goto n
  - op_a ::= + | - | * | / | …
  - op_r ::= < | ≤ | = | > | ≥ | ...
  - P ∈ **Num** → *I*

While3Addr Extensions (more later)

`I ::= x := n | x := y | x := y op z | goto n | if x` op_r `0 goto n`

`| x := f(y)`

`| return x`

`| x := y.m(z)`

`| read x`

`| print x`

`| x := &p`

`| x := *p`

`| *p := x`

`| x := y.f`

`| x.f := y`

WHILE3ADDR Semantics

- Configuration (state) includes environment + program counter:

$$c \in E \times \mathbb{N}$$

- Evaluation occurs with respect to a global program that maps labels to instructions: $P \in \mathbb{N} \rightarrow I$

$$P \vdash < E, n > \rightsquigarrow < E', n' >$$

(c) J. Aldrich, C. Le Goues, R. Padhye

$$\frac{P(n) = x := m}{P \vdash \langle E, n \rangle \leadsto \langle E[x \mapsto m], n+1 \rangle}\ \textit{step-const}$$

$$\frac{P[n] = x := y}{P \vdash \langle E, n \rangle \leadsto \langle E[x \mapsto E(y)], n+1 \rangle}\ \textit{step-copy}$$

$$\frac{P(n) = x := y\ op\ z \quad E(y)\ \mathbf{op}\ E(z) = m}{P \vdash \langle E, n \rangle \leadsto \langle E[x \mapsto m], n+1 \rangle}\ \textit{step-arith}$$

$$\frac{P(n) = \text{goto}\ m}{P \vdash \langle E, n \rangle \leadsto \langle E, m \rangle}\ \textit{step-goto}$$

$$\frac{P(n) = \text{if}\ x\ op_r\ 0\ \text{goto}\ m \quad E(x)\ \mathbf{op_r}\ 0 = true}{P \vdash \langle E, n \rangle \leadsto \langle E, m \rangle}\ \textit{step-iftrue}$$

$$\frac{P(n) = \text{if}\ x\ op_r\ 0\ \text{goto}\ m \quad E(x)\ \mathbf{op_r}\ 0 = false}{P \vdash \langle E, n \rangle \leadsto \langle E, n+1 \rangle}\ \textit{step-iffalse}$$