

# Dcheck: A Derivation Checker

Karl Crary

November 11, 2021

## 1 Overview

Dcheck provides a simple, text-based framework for writing and checking logic derivations. Consider the (blackboard-style) natural-deduction derivation:

$$\frac{\frac{\overline{A \text{ true}}^u \quad \overline{A \text{ true}}^u}{A \wedge A \text{ true}} \wedge I}{A \supset (A \wedge A) \text{ true}} \supset I^u$$

In Dcheck the same derivation is written:

```
system ND
deriv simple =
  A => (A /\ A) true
  by ImpI(u)
  >>
  A /\ A true
  by AndI
  >>
  {
    A true
    by u
  }

  {
    A true
    by u
  }
```

The first line indicates that the system we are working in is natural deduction (“ND”). Following that is the derivation. In it, we can see one of the main ways in which Dcheck derivations differ from blackboard derivations (apart from being written in ASCII): Dcheck derivations grow downward, not upward.

A derivation consists of:

- a judgement (*e.g.*, `A => (A /\ A) true`),

- the keyword “by” followed by a *reason*, which is usually a rule or a hypothesis name (*e.g.*, `ImpI(u)` or `u`), and
- zero or more premises, each of which is a derivation.

If a rule has one or more premises, the symbol “>>” separates the reason from the premises. If a rule has two or more premises, each premise must be enclosed in curly braces. If a rule has one premise, the braces are optional (in the example above they are omitted).

A Dcheck program is a sequence of clauses, each one of either:

- defines a derivation, written `deriv <name> = <derivation>`
- defines a proposition abbreviation, written `prop <name> = <proposition>`.
- sets the current logical system, written `system <system-name>`.

Comments can be included using the SML comment convention (that is, `(* ignored text *)`).

## 2 Propositions and Judgements

The syntax of propositions is given in the following table, in decreasing order of precedence:

connective	blackboard	Dcheck
truth	$T$	<b>T</b>
falsity	$F$	<b>F</b>
positive truth	$T^+$	<b>T+</b>
negative truth	$T^-$	<b>T-</b>
not <sup>1</sup>	$\neg$	<b>~</b>
upshift	$\uparrow$	<b>up</b>
downshift	$\downarrow$	<b>down</b>
and	$\wedge$	<b>/\</b>
positive and	$\wedge^+$	<b>/\+</b>
negative and	$\wedge^-$	<b>/\-</b>
or	$\vee$	<b>\/</b>
implies	$\supset$	<b>=&gt;</b>

Any upper-case proposition identifier (other than **T** or **F**) is taken to be a metavariable. In systems where atomicity matters (*e.g.*, sequent calculus), any metavariable beginning with the letter **P**, **Q**, **R**, or **S** is taken to be atomic. Any lower-case proposition identifier refers to a proposition abbreviation that was defined earlier (for example, by the clause `prop t_and_t = T /\ T`). In focused logic, a metavariable should end with a plus or minus to indicate polarity (such as **P-** for a negative atomic proposition).

The syntax of judgements is given in the following table:

<sup>1</sup>This is the defined not ( $\neg P = P \supset F$ ) unless the current system is classical, in which case it is classical logic's primitive not.

system (system-name)	blackboard	Dcheck
natural deduction (ND)	$A \text{ true}$	$A \text{ true}$
sequent calculus (SC)	$A_1, \dots, A_n \implies B$	$A1, \dots, An \implies B$
classical logic (CL)	$A \text{ true}$	$A \text{ true}$
	$A \text{ false}$	$A \text{ false}$
	$\#$	$\#$
focused logic (FL)	$A_1, \dots, A_n; B_1, \dots B_m \xrightarrow{R} C$	$A1, \dots, An ; B1, \dots, Bm \text{-r-} \rightarrow C$
	$A_1, \dots, A_n; B_1, \dots B_m \xrightarrow{L} C$	$A1, \dots, An ; B1, \dots, Bm \text{-l-} \rightarrow C$
	$A_1, \dots, A_n \longrightarrow C$	$A1, \dots, An \text{->} C$
	$A_1, \dots, A_n; [B] \longrightarrow C$	$A1, \dots, An ; [B] \text{->} C$
	$A_1, \dots, A_n \longrightarrow [C]$	$A1, \dots, An \text{->} [C]$

Additional systems may be supported in the future.

### 3 Rules and Reasons

The rule sets of natural deduction, sequent calculus, classical logic<sup>2</sup>, and focused logic are given in Figures 1–4. The name of a defined derivation (for example, by the clause `deriv foo = A /\ B true by ...`) or a hypothesis can also be used as a reason.

An important difference between blackboard derivations and Dcheck derivations is **Dcheck premises must be given in the standard order**. For example, in the following fragment of a derivation, the two premises `A true` and `B true` *cannot* be given in the opposite order (whereas in a blackboard derivation the order would not matter):

```
...
A /\ B true
by AndI
>>
{
  A true
  by ...
}

{
  B true
  by ...
}
```

Additionally, some rules require an assumption number (notably sequent-calculus left rules). In such rules, assumptions are counted from right to left (with the rightmost being 0). For example:

---

<sup>2</sup>or, more precisely, classical natural deduction

```

system SC
deriv another_simple =
  ==> P /\ Q => P
  by ImpR
  >>
  P /\ Q ==> P
  by AndL1(0)
  >>
  P /\ Q, P ==> P
  by AndL2(1)
  >>
  P /\ Q, P, Q ==> P
  by Init(1)

```

As usual in sequent calculus, assumptions are taken to be unordered. Also, unneeded assumptions can be silently dropped. For example, the following derivation fragment is legal:

```

...
A /\ B, C, D ==> E
by AndL1(2)
>>
D, A, C ==> E
by ...

```

This also applies to focused logic, except assumptions in the stoup<sup>3</sup> cannot be reordered or dropped.

## 4 Using the checker, and additional resources

When you submit your solution to Gradescope, the autograder will first run a set of sanity checks. These ensure that your solution parses correctly and passes some other elementary checks. If your solution passes the sanity checks, the autograder will grade it and produce output for any problems with instant feedback. The full results will be visible when the assignment is over.

- You can run the sanity checks by themselves on Andrew by executing `~crary/bin/dsanity <filename>`.
- You can visualize your program in blackboard-style structure (*i.e.*, derivations growing upward, horizontal lines to separate premises from conclusion) by running `~crary/bin/dvis <filename>`.
- There is a set of examples at `cs.cmu.edu/~crary/317-f21/example.deriv`.

---

<sup>3</sup>the second group of assumptions in inversion stages

blackboard	Dcheck
$\wedge I$	<b>AndI</b>
$\wedge E1$	<b>AndE1</b>
$\wedge E2$	<b>AndE2</b>
$\supset I$	<b>ImpI</b> (⟨name⟩)
$\supset E$	<b>ImpE</b>
$\vee I1$	<b>OrI1</b>
$\vee I2$	<b>OrI2</b>
$\vee E$	<b>OrE</b> (⟨name⟩, ⟨name⟩)
$TI$	<b>TI</b>
$FE$	<b>FE</b>

Figure 1: Natural Deduction (ND) Rules

blackboard	Dcheck
<i>Init</i>	<b>Init</b> (⟨number⟩)
$\wedge R$	<b>AndR</b>
$\wedge L1$	<b>AndL1</b> (⟨number⟩)
$\wedge L2$	<b>AndL2</b> (⟨number⟩)
$\supset R$	<b>ImpR</b>
$\supset L$	<b>ImpL</b> (⟨number⟩)
$\vee R1$	<b>OrR1</b>
$\vee R2$	<b>OrR2</b>
$\vee L$	<b>OrL</b> (⟨number⟩)
$TR$	<b>TR</b>
$FL$	<b>FL</b> (⟨number⟩)

Figure 2: Sequent Calculus (SC) Rules

blackboard	Dcheck
$\wedge T$	<b>AndT</b>
$\wedge F1$	<b>AndF1</b>
$\wedge F2$	<b>AndF2</b>
$\supset T$	<b>ImpT</b> (⟨name⟩)
$\supset F$	<b>ImpF</b>
$\vee T1$	<b>OrT1</b>
$\vee T2$	<b>OrT2</b>
$\vee F$	<b>OrF</b>
$TT$	<b>TT</b>
$FF$	<b>FF</b>
$\neg T$	<b>NotT</b>
$\neg F$	<b>NotF</b>
$T\#$	<b>ContraT</b> (⟨name⟩)
$F\#$	<b>ContraF</b> (⟨name⟩)
$\#$	<b>Contra</b>

Figure 3: Classical Logic (CL) Rules

blackboard	Dcheck
$PR$	PR
$\uparrow R$	UpR
$\supset R$	ImpR
$\wedge^- R$	AndmR
$T^- R$	TmR
$PL$	PL
$\downarrow L$	DownL
$\wedge^+ L$	AndpL
$T^+ L$	TpL
$\vee L$	OrL
$FL$	FL
$Stable$	Stable
$FocusL$	FocusL( $\langle \text{number} \rangle$ )
$FocusR$	FocusR
$Init^-$	Initm
$\uparrow L$	UpL
$\supset L$	ImpL
$\wedge^- L1$	AndmL1
$\wedge^- L2$	AndmL2
$Init^+$	Initp( $\langle \text{number} \rangle$ )
$\downarrow R$	DownR
$\wedge^+ R$	AndpR
$T^+ R$	TpR
$\vee R1$	OrR1
$\vee R2$	OrR2

Figure 4: Focused Logic (FL) Rules