

Dcheck: A Derivation Checker

Karl Crary

October 7, 2021

1 Overview

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

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

In Dcheck the same derivation is written:

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

  {
    A true
    by u
  }
```

In the example, 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 which is:

- a definition of a derivation, written `deriv <name> = <derivation>`
- a definition of a proposition abbreviation, written `prop <name> = <proposition>`.

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	T
falsity	F	F
not	\neg	~
and	\wedge	\&
or	\vee	\
implies	\supset	=>

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`).

The syntax of judgements is given in the following table:

system	blackboard	Dcheck
natural deduction	$A \text{ true}$	A true
sequent calculus	$A_1, \dots, A_n \implies B$	A1, ..., An ==> B

Additional systems may be supported in the future.

3 Rules and Reasons

The rule sets of natural deduction and sequent calculus are given in Figures 1–2. 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:

```
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)
```

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

Figure 1: Natural Deduction Rules

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

Figure 2: Sequent Calculus Rules

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 ...

```

4 Using the checker

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>`.