Dcheck: A Derivation Checker

Karl Crary

February 7, 2022

1 Overview

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

$$\cfrac{\cfrac{\cfrac{}{A\ \mathsf{true}}\,u \qquad \cfrac{}{A\ \mathsf{true}}\,u}{A \wedge A\ \mathsf{true}}\,\wedge I}{A \supset (A \wedge A)\ \mathsf{true}}\,\supset I^u$$

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. Each group has greater precedence than the groups below it. (Don't worry if you aren't familiar with all these connectives yet.)

connective	blackboard	Dcheck
truth	T	`T`
falsity	F	`F`
positive truth	T^+	`T+`
negative truth	T^-	`T-`
top	$\top$	`T`
one	1	`1`
zero	0	`0`
equality	$=$	`=`
not[1]	$\neg$	`~`
upshift	$\uparrow$	`up`
downshift	$\downarrow$	`down`
bang	$!$	`!`
box	$\Box$	`[]`
diamond	$\Diamond$	`<>`
and	$\wedge$	`/\`
positive and	$\wedge^+$	`/\+`
negative and	$\wedge^-$	`/\-`
tensor	$\otimes$	`*`
or	$\vee$	`\/`
plus	$\oplus$	`+`
with	$\&$	`&`
implies	$\supset$	`=>`
lolli	$\multimap$	`-o`

[1]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.

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). In predicate logic (*e.g.,* Heyting arithmetic), a predicate applied to a term is written `A(n)`.

The syntax of judgements is given in the following table:

system (system-name)	blackboard	Dcheck
natural deduction (`ND`)	$A\ \mathsf{true}$	`A true`
natural deduction with contexts (`NDC`)	$A_1\ \mathsf{true}, ..., A_n\ \mathsf{true} \vdash B\ \mathsf{true}$	`A1 true, ..., An true |- B true`
verifications & uses (`VU`)	$A\uparrow$	`A ver`
	$A\downarrow$	`A use`
Heyting arithmetic (`AR`)	$A\ \mathsf{true}$	`A true`
sequent calculus (`SC`)	$A_1, \ldots, A_n \Longrightarrow B$	`A1, ..., An ==> B`
classical logic (`CL`)	$A\ \mathsf{true}$	`A true`
	$A\ \mathsf{false}$	`A false`
	$\#$	`#`
focused logic (FL)	$A_1, \ldots, A_n; B_1, \ldots B_m \xrightarrow{R} C$	`A1,..., An ; B1,..., Bm -r-> C`
	$A_1, \ldots, A_n; B_1, \ldots B_m \xrightarrow{L} C$	`A1,..., An ; B1,..., Bm -l-> C`
	$A_1, \ldots, A_n \longrightarrow C$	`A1,..., An --> C`
	$A_1, \ldots, A_n; [B] \longrightarrow C$	`A1,..., An ; [B] --> C`
	$A_1, \ldots, A_n \longrightarrow [C]$	`A1,..., An --> [C]`
linear logic (`LL`)	$A_1\ \mathsf{valid}, ..., A_m\ \mathsf{valid}; B_1\ \mathsf{true}, ..., B_n\ \mathsf{true} \Vdash C\ \mathsf{true}$	`A1 valid, ..., Am valid, B1 true, ..., Bn true |- C true`
modal logic (`ML`)	$A_1\ \mathsf{valid}, ..., A_m\ \mathsf{valid}; B_1\ \mathsf{true}, ..., B_n\ \mathsf{true} \vdash C\ \mathsf{true}$	`A1 valid, ..., Am valid, B1 true, ..., Bn true |- C true`
	$A_1\ \mathsf{valid}, ..., A_m\ \mathsf{valid}; B_1\ \mathsf{true}, ..., B_n\ \mathsf{true} \vdash C\ \mathsf{poss}$	`A1 valid, ..., Am valid, B1 true, ..., Bn true |- C poss`

3 Rules and Reasons

The rule sets of the various systems are given in Figures 1–7. Each rule may be used as a reason. 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 assumption **zero**). For example:

```
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)
```

In modal logic, where multiple sorts of hypothesis can appear in the context, a hypothesis's index is based only on hypotheses of the same sort. For example, if the context is `A valid, B true`, the location of the `A` hypothesis is validity hypothesis zero, not as (overall) hypothesis one. (Multiple sorts of hypotheses can also appear in the context in linear logic, but it turns out this counting issue never arises. Can you see why?)

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[2] cannot be reordered or dropped.

When quantifiers have introduced a term parameter, say $a : T$, the judgement `a : T` can be derived using `a` as the reason. For example:

[2]the second group of assumptions in inversion stages

```
deriv all_expand =
   (All x:T. A(x)) => (All x:T. A(x)) true
   by ImpI(u)
   >>
   All x:T. A(x) true
   by AllI(a)
   >>
   A(a) true
   by AllE
   >>
      {
      All x:T. A(x) true
      by u
      }

      {
      a : T
      by a
      }
```

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>`. (Note that the visualizer only visualizes the derivation; it does not run any sanity checks.)
- There is a set of examples at `cs.cmu.edu/~crary/dcheck/example.deriv`.

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 (`ND`) Rules

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

Figure 2: Natural Deduction with Contexts (`NDC`) Rules

blackboard	Dcheck
$\wedge\uparrow$	`AndI`
$\wedge\downarrow 1$	`AndE1`
$\wedge\downarrow 2$	`AndE2`
$\supset\uparrow$	`ImpI`(⟨name⟩)
$\supset\downarrow$	`ImpE`
$\vee\uparrow 1$	`OrI1`
$\vee\uparrow 2$	`OrI2`
$\vee\downarrow$	`OrE`(⟨name⟩, ⟨name⟩)
$T\uparrow$	`TI`
$F\downarrow$	`FE`
$\downarrow\uparrow$	`UV`

Figure 3: Verifications and Uses (`VU`) Rules

blackboard	Dcheck
$\wedge I$	`AndI`
$\wedge E1$	`AndE1`
$\wedge E2$	`AndE2`
$\supset I$	`ImpI`($\langle$name$\rangle$)
$\supset E$	`ImpE`
$\vee I1$	`OrI1`
$\vee I2$	`OrI2`
$\vee E$	`OrE`($\langle$name$\rangle$, $\langle$name$\rangle$)
TI	`TI`
FE	`FE`
$\forall I$	`AllI`($\langle$name$\rangle$)
$\forall E$	`AllE`
$\exists I$	`ExistI`
$\exists E$	`ExistE`($\langle$name$\rangle$, $\langle$name$\rangle$)
$\mathtt{nat}I_0$	`NatI0`
$\mathtt{nat}I_s$	`NatIs`
$\mathtt{nat}E$	`NatE`($\langle$name$\rangle$, $\langle$name$\rangle$)
$=I_{00}$	`EqI00`
$=I_{ss}$	`EqIss`
$=E_{ss}$	`EqEss`
$=E_{0s}$	`EqE0s`
$=E_{s0}$	`EqEs0`

Figure 4: Heyting Arithmetic (`AR`) Rules

blackboard	Dcheck
$\wedge T$	`AndT`
$\wedge F1$	`AndF1`
$\wedge F2$	`AndF2`
$\supset T$	`ImpT`($\langle$name$\rangle$)
$\supset F$	`ImpF`
$\vee T1$	`OrT1`
$\vee T2$	`OrT2`
$\vee F$	`OrF`
TT	`TT`
FF	`FF`
$\neg T$	`NotT`
$\neg F$	`NotF`
$T\#$	`ContraT`($\langle$name$\rangle$)
$F\#$	`ContraF`($\langle$name$\rangle$)
$\#$	`Contra`

Figure 5: Classical Logic (`CL`) Rules

blackboard	Dcheck
Init	`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 6: Sequent Calculus (`SC`) 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`(⟨number⟩)
FocusR	`FocusR`
Init^-	`Initm`
$\uparrow L$	`UpL`
$\supset L$	`ImpL`
$\wedge^- L1$	`AndmL1`
$\wedge^- L2$	`AndmL2`
Init^+	`Initp`(⟨number⟩)
$\downarrow R$	`DownR`
$\wedge^+ R$	`AndpR`
$T^+ R$	`TpR`
$\vee R1$	`OrR1`
$\vee R2$	`OrR2`

Figure 7: Focused Logic (`FL`) Rules

blackboard	Dcheck
Hyp	`Hyp`
Hypv	`Hypv(`⟨number⟩`)`
$\otimes I$	`TensI`
$\otimes E$	`TensE`
$\& I$	`WithI`
$\& E1$	`WithE1`
$\& E2$	`WithE2`
$\oplus I1$	`PlusI1`
$\oplus I2$	`PlusI2`
$\oplus E$	`PlusE`
$\multimap I$	`LolI`
$\multimap E$	`LolE`
$!I$	`BangI`
$!E$	`BangE`
$\top I$	`TopI`
1I	`OneI`
1E	`OneE`
0E	`ZeroE`

Figure 8: Linear Logic (`LL`) Rules

blackboard	Dcheck
Hyp	`Hyp(`⟨number⟩`)`
Hypv	`Hypv(`⟨number⟩`)`
$\wedge I$	`AndI`
$\wedge E1$	`AndE1`
$\wedge E2$	`AndE2`
$\supset I$	`ImpI`
$\supset E$	`ImpE`
$\vee I1$	`OrI1`
$\vee I2$	`OrI2`
$\vee E$	`OrE`
TI	`TI`
FE	`FE`
$\Box I$	`BoxI`
$\Box E$	`BoxE`
$\Box E_p$	`BoxEp`
$\Diamond I$	`DiaI`
Here	`Here`
$\Diamond E$	`DiaE`

Figure 9: Modal Logic (`ML`) Rules