PTcalc Manual

Kokichi Futatsugi

September 3, 2021

1 Overview

A CafeOBJ's specification (i.e. a module) M contains equations and defines the set of constructor models $\mathbb{M}od(M)$ each element of that satisfies the equations. PTcalc (Proof Tree Calculus) is a refined version of a CafeOBJ version of CITP (Constructor-based Inductive Theorem Prover) and helps to prove a property holds for any model in $\mathbb{M}od(M)$. A property is described as a Boolean ground term p with fresh constants that correspond to the parameters of the property.

If the term p is deduced to be equal to `true` with the M's equations (in symbols $M \vdash_{\mathrm{e}} p$), any model of M satisfies p (in symbols $M \models p$) by the soundness of equational deduction (in symbols $M \vdash_{\mathrm{e}} p \Rightarrow M \models p$). If the term p is reduced to `true` by using the M's equations as reduction (rewriting) rules from left to right (in symbols $M \vdash_{\mathrm{r}} p$), $M \vdash_{\mathrm{e}} p$ holds by the soundness of reduction with respect to equational deduction (in symbols $M \vdash_{\mathrm{r}} p \Rightarrow M \vdash_{\mathrm{e}} p$). Let $M \vdash_{\mathrm{c}} p$ denote that the CafeOBJ reduction command "`red in` M `:` p `.`" returns `true`. CafeOBJ's reduction is an implementation of $M \vdash_{\mathrm{r}} p$ and we get $M \vdash_{\mathrm{c}} p \Rightarrow M \vdash_{\mathrm{r}} p$. Because implication ($\Rightarrow$) is transitive the following proof rule is obtained.

(1) $\quad M \vdash_{\mathrm{c}} p \Rightarrow M \models p$

Usually $M \vdash_{\mathrm{c}} p$ is difficult to prove directly, and we need to find case splitting equations $e_1, \cdots, e_n$ such that at least one of them holds for any model in $\mathbb{M}od(M)$. That is, the equations cover all the possibilities (i.e. are exhaustive). Let M_{+e_i} be the module gotten by adding e_i to M, then each model in $\mathbb{M}od(M)$ is a model of M_{+e_j} for some $j \in \{1, 2, \cdots, n\}$, and we get the following proof rule of **case split with exhaustive equations**.

(2) $\quad (M_{+e_1} \models p \wedge M_{+e_2} \models p \wedge \cdots \wedge M_{+e_n} \models p) \Rightarrow M \models p$

$M_{+e_i} \vdash_{\mathrm{c}} p$ would be still difficult to prove and **(2)** is applied repeatedly. The repeated applications of **(2)** generate **proof trees** successively. Each of the generated proof trees has the **root node** $M \models p$ and each of other **nodes** is of the form $M_{+e_{i_1} \cdots + e_{i_m}} \models p$ $(m \in \{1, 2, \cdots\})$ that is generated as a **child node** of $M_{+e_{i_1} \cdots + e_{i_{m-1}}} \models p$ by applying **(2)**. A **leaf node** (i.e. a node without child nodes) $M_{+e_{l_1}\cdots+e_{l_k}} \models p$ ($k \in \{0, 1, \cdots\}$) of a proof tree is called **effective** if $M_{+e_{l_1}\cdots+e_{l_k}} \models_{\mathrm{c}} p$ holds. A proof tree is called effective if all of whose leaf nodes are effective. PTcalc proves $M \models p$ by constructing an effective proof tree whose root node is $M \models p$.

2 PTcalc Commands

Each of the PTcalc commands starts with the keyword `:goal`, `:def`, `:apply`, `:csp`, `:ctf`, `:init`, `:red`, `:show`, `:describe`, `:select`, or `:set`. The head character `:` distinguishes them from ordinary CafeOBJ commands.

In PTcalc, a node in a proof tree is a special CafeOBJ module and called a **goal**. Each goal *gl* has a **name** like `root` or `1-2-3` (i.e. 3rd child of 2nd child of 1st child of `root`), and consists of the following five items.

(1) The **next target (or default) goal** Boolean tag NTG(*gl*) that indicates a goal where a PTcalc command is executed. (NTG(*gl*) = `true`) holds for at most one goal *gl* in a proof tree.

(2) The **context module** CTM(*gl*) that is a CafeOBJ module and correspoonts to M of $M \models p$. The goal *gl* inherits (imports) all the contents of CTM(*gl*).

(3) The set of **introduced axioms (i.e. assumptions)** INA(*gl*) that corresponds to $+e_{i_1} \cdots + e_{i_m}$ of $M_{+e_{i_1}\cdots+e_{i_m}} \models p$.

(4) The set of **sentences (or equations) to be proved** STP(*gl*) that corresponts to p of $M_{+e_{i_1}\cdots+e_{i_m}} \models p$.

(5) The **discharged** Boolean tag DCD(*gl*) that indicates whether *gl* is already discharged (i.e. proved).

(CTM(*gl*)∪INA(*gl*))⊨STP(*gl*) corresponds to $M_{+e_{i_1}\cdots+e_{i_m}} \models p$, where (CTM(*gl*)∪ INA(*gl*)) is understood as the module gotten by adding all the equations in INA(*gl*) to CTM(*gl*), and STP(*gl*) is understood as the conjunction of its elements. *gl* sometimes means (CTM(*gl*)∪INA(*gl*)).

The effects of PTcalc command executions are defined in the following subsections where a succession of commands ":$cmnd_1$:$cmnd_2$" means executing :$cmnd_2$ after executing :$cmnd_1$.

2.1 :goal

For $n \in \{1, 2, \cdots\}$ unconditional equations eq_i ($i \in \{1, 2, \cdots, n\}$) a `:goal` command

:goal{eq_l eq_2 $\cdots$ eq_n}

initiates a proof by generating a proof tree that consists only of the goal `root` as follows.

(1) NTG(`root`) = `true`.
(2) CTM(`root`) is the **current module** of CafeOBJ.
(3) INA(`root`) = {} (i.e. the empty set).
(4) STP(`root`) = $\{eq_l, eq_2, \cdots, eq_n\}$.
(5) DCD(`root`) = `false`.

A CafeOBJ module M becomes the current module by executing a CafeOBJ command "`select` M `.`". If some contents (i.e. sorts, operators, equations) are declared after a CafeOBJ command "`open` M `.`", the `open`ed tentative module `%` obtained by adding the contents to M becomes the current module.

2.2 `:def`

A `:def` command gives a name to a `:csp` command, a `:init` command, or a sequence `(`$\cdots$`)` of command names as follows.

`:def` *csid* `= :csp`$\cdots$

`:def` *inid* `= :init`$\cdots$

`:def` *sqid* `= (`$id_1\ id_2 \cdots id_n$`)` $\quad (n \in \{1, 2, \cdots\})$

Each id_i $(i \in \{1, 2, \cdots, n\})$ is the built-in command name `rd-` or `rd`, or the already defined `:csp` command name *csid*, `:init` command name *inid*, or `(`$\cdots$`)` command name *sqid*.

2.3 `:apply(`*csid*`)` and `:csp{`$\cdots$`}` and `:ctf{eq` l `=` r `.}`

Let (a) *tg* be a goal such that (`NTG(`*tg*`)` $=$ `true`) and (b) *csid* be the name of a `:csp` command defined as follows with $n \in \{1, 2, \cdots\}$ equations eq_i $(i \in \{1, 2, \cdots, n\})$.

`:def` *csid* `= :csp{`$eq_1\ \ eq_2\ \cdots\ eq_n$`}`

Then executing the command

`:apply(`*csid*`)`

generates n sub-goals (i.e. child goals) *tg*`-1`, *tg*`-2`, $\cdots$, *tg*`-n` of *tg* as follows.

(1:1) Change `NTG(`*tg*`)` from `true` to `false`.

(1:2) `NTG(`*tg*`-1)` $=$ `true`.

(1:3) `NTG(`*tg*`-`i`)` $=$ `false` $(i \in \{2, \cdots, n\})$.

(2) `CTM(`*tg*`-`i`)` $=$ `CTM(`*tg*`)` $\quad (i \in \{1, 2, \cdots, n\})$.

(3) `INA(`*tg*`-`i`)` $=$ `INA(`*tg*`)`$\cup\{eq_i\}$ $\quad (i \in \{1, 2, \cdots, n\})$.

(4) `STP(`*tg*`-`i`)` $=$ `STP(`*tg*`)` $\quad (i \in \{1, 2, \cdots, n\})$.

(5) `DCD(`*tg*`-`i`)` $=$ `false` $\quad (i \in \{1, 2, \cdots, n\})$.

Executing the `:csp` command

`:csp{`$eq_1\ \ eq_2\ \cdots\ eq_n$`}`

without giving a name has the same effect as defined above.

`:ctf{eq` l `=` r `.}` is the abbreviation of

`:csp{eq` l `=` r `.  eq (`l `=` r`) = false .}`.

2.4 `:apply(`*inid*`)` and "`:init as` *eqid* `[` *lb* `] by {` *sbst* `}`"

Let (a) *tg* be a goal such that (`NTG(`*tg*`)` $=$ `true`), (b) *lb* be a label of an equation in *tg* (i.e. in `CTM(`*tg*`)`$\cup$`INA(`*tg*`)`) with a set of labels *lbset* that includes *lb*, (c) *inid* be the name of a `:init` command defined as follows with a new equation name *eqid* and a substitution *sbst*, where *sbst* is a sequence of $n \in \{1, 2, \cdots\}$ variable-term pairs "v_i `<-` t_i `;`" $(i \in \{1, 2, \cdots, n\})$.

`:def` *inid* `= :init as` *eqid* `[` *lb* `] by {` *sbst* `}` "`as` *eqid*" can be omitted, `[` *lb* `]` can be replaced with `(` *eq* `)` by writing *eq* directly, and "by { *sbst* }" can be omitted by replacing it with "`.`". Let "`ceq` l `=` r `if` c `.`" be the equation with the label *lb* or *eq* (`ceq` stands for conditional equation). An unconditional equation is understood to have the condition `true`, i.e., $c =$ `true`.

Then executing the command

`:apply(`*inid*`)`

generates a sub-goal *tg*-`1` of the goal *tg* as follows, where $\widehat{l}$, $\widehat{r}$, $\widehat{c}$ are normal forms of *sbst*(l), *sbst*(r), *sbst*(c) in the goal *tg*. If "by { *sbst* }" is omitted, *sbst* is understood to be the identity function. The equation "`ceq` l `=` r `if` c `.`" itself is not used to get the normal forms if it is declared with `(...)` (i.e. it is *eq*). The equation is, however, used if it is the equation with the label *lb* and without `:nonexec` attribute, hence the `:nonexec` attribute should be declared if the label *lb* is supposed to be used in a `:init` command. Note that, if the condition c is `true`, using "`ceq` l `=` r `if` c `.`" for getting $\widehat{l}$, $\widehat{r}$, $\widehat{c}$ generates a trivial equation "`ceq` $\widehat{r}$ `=` $\widehat{r}$ `if true .`" i.e., "`eq` $\widehat{r}$ `=` $\widehat{r}$ `.`" (see 3:1 below).

(1:1) Change `NTG`(*tg*) from `true` to `false`.

(1:2) `NTG`(*tg*-`1`) = `true`.

(2) `CTM`(*tg*-`1`) = `CTM`(*tg*).

(3:0) `INA`(*tg*-`1`) = `INA`(*tg*).

(4) `STP`(*tg*-`1`) = `STP`(*tg*).

(5) `DCD`(*tg*-`1`) = `false`.

(3:1) Add "`ceq[`*eqid* `:nonexec]:` $\widehat{l}$ `=` $\widehat{r}$ `if` $\widehat{c}$ `.`" to `INA`(*tg*-`1`)
if ($\widehat{c}=$ `false`)$\vee$($\widehat{l}=\widehat{r}$).

(3:2) Add "`eq[`*eqid*`]:` $\widehat{r}$ `=` $\widehat{l}$ `.`" to `INA`(*tg*-`1`)
if ($\widehat{c}=$ `true`)$\wedge\neg$($\widehat{l}=\widehat{r}$)$\wedge$(($\widehat{l}=$ `true`)$\vee$($\widehat{l}=$ `false`)).

(3:3) Add "`eq[`*eqid*`]:` $\widehat{l}$ `=` $\widehat{r}$ `.`" to `INA`(*tg*-`1`)
if ($\widehat{c}=$ `true`)$\wedge\neg$($\widehat{l}=\widehat{r}$)$\wedge\neg$(($\widehat{l}=$ `true`)$\vee$($\widehat{l}=$ `false`)).

(3:4) Add "`ceq[`*eqid*`]:` $\widehat{r}$ `=` $\widehat{l}$ `if` $\widehat{c}$ `.`" to `INA`(*tg*-`1`)
if $\neg$(($\widehat{c}=$ `true`)$\vee$($\widehat{c}=$ `false`))$\wedge\neg$($\widehat{l}=\widehat{r}$)$\wedge$
(($\widehat{l}=$ `true`)$\vee$($\widehat{l}=$ `false`)).

(3:5) Add "`ceq[`*eqid*`]:` $\widehat{l}$ `=` $\widehat{r}$ `if` $\widehat{c}$ `.`" to `INA`(*tg*-`1`)
if $\neg$(($\widehat{c}=$ `true`)$\vee$($\widehat{c}=$ `false`))$\wedge\neg$($\widehat{l}=\widehat{r}$)$\wedge$
$\neg$(($\widehat{l}=$ `true`)$\vee$($\widehat{l}=$ `false`)).

Executing the `:init` command

`:init as` *eqid* `[` *lb* `] by {` *sbst* `}`

without giving a name has the same effect as defined above.

2.5 `:apply(rd-)`

Let (a) *tg* be the goal such that (`NTG`(*tg*) = `true`) and (b) `STP`(*tg*) = {eq_1, $eq_2, \cdots, eq_n$} ($n \in \{1, 2, \cdots\}$) with l_i and r_i being the left hand and the right hand of eq_i ($i \in \{1, \cdots, n\}$). Let a goal *gl-s* be a **sibling goal** of a goal *gl* if *gl-s* and *gl* are child goals of the same goal.

Then executing the command

`:apply(rd-)`

changes STP(tg), DCD(_), NTG(_) with the following procedure. If INA(tg) contains executable (i.e. not having `:nonexec` attribute) "`eq true = false .`" or "`eq false = true .`", the **contradictory** equation has been generated with the equations in CTM(tg)∪INA(tg) in the execution of `:init` command (see 2.4) and the goal tg can be discharged.

```
IF (INA(tg) does not contain executable
          "eq true = false ." or "eq false = true .")
THEN
   [ For i ∈ {1,···,n} let ^eq_i be the normal form of (l_i = r_i) in tg ] ;
   [ (4) For i ∈ {1,···,n} erase eq_i from STP(tg) if (^eq_i = true) ] ;
   IF not(STP(tg) = {}) THEN STOP FI
FI ;
[ (5:1) Change DCD(tg) to true ] ;
[ (1:1) Change NTG(tg) to false ] ;
[ Let tmp be tg ] ;
WHILE ((DCD(tmp-s) = true) for any sibling goal tmp-s of tmp)
DO
   [ (5:2) Change DCD(tmp-p) to true for the parent goal tmp-p of tmp ] ;
   [ Let tmp be tmp-p ]
OD ;
[ (1:2) Change NTG(lf) to true for the goal lf that is the first leaf goal
       in the lexicographic order such that (DCD(lf) = false)
       if such goal lf exists ]
```

2.6 :apply(rd)

Let (a) tg be the goal such that (NTG(tg) = `true`) and (b) STP(tg) = $\{eq_1, eq_2, \cdots, eq_n\}$ ($n \in \{1, 2, \cdots\}$) with l_i and r_i being the left hand and the right hand of eq_i ($i \in \{1, \cdots, n\}$).

Then executing the command

`:apply(rd)`

generates a sub-goal tg-1 of the goal tg and changes STP(tg-1), DCD(_), NTG(_) as follows.

(1:1) Change NTG(tg) from `true` to `false`.
(1:2) NTG(tg-1) = `true`.
(2) CTM(tg-1) = CTM(tg).
(3) INA(tg-1) = INA(tg).
(4:1) STP(tg-1) = STP(tg).
(5:1) DCD(tg-1) = `false`.

```
IF (INA(tg-1) does not contain executable
          "eq true = false ." or "eq false = true .")
THEN
   [ For i ∈ {1,···,n} let ^l_i, ^r_i, ^eq_i be the normal forms
         of l_i, r_i, (l_i = r_i) in tg-1 ] ;
   [ (4:2) For each eq_i ∈ STP(tg-1) (i ∈ {1,···,n})
         if (^eq_i = true) then erase eq_i
                      else replace eq_i with "eq ^l_i = ^r_i ." fi ] ;
```

IF not($\texttt{STP}(tg\texttt{-1}) = \{\}$) **THEN STOP FI**
FI ;
[(5:2) Change $\texttt{DCD}(tg\texttt{-1})$ to `true`] **;**
[(1:3) Change $\texttt{NTG}(tg\texttt{-1})$ to `false`] **;**
[Let *tmp* be $tg\texttt{-1}$] **;**
WHILE (($\texttt{DCD}(tmp\text{-}s) = \texttt{true}$) for any sibling goal *tmp-s* of *tmp*)
DO
[(5:3) Change $\texttt{DCD}(tmp\text{-}p)$ to `true` for the parent goal *tmp-p* of *tmp*] **;**
[Let *tmp* be *tmp-p*]
OD ;
[(1:4) Change $\texttt{NTG}(lf)$ to `true` for the goal *lf* that is the first leaf goal
in the lexicographic order such that ($\texttt{DCD}(lf) = \texttt{false}$)
if such goal *lf* exists]

2.7 `:select`

Let *gl* and *tg* be goals such that ($\texttt{NTG}(tg) = \texttt{true}$). The command
`:select` *gl* .
changes $\texttt{NTG}(tg)$ to `false` and $\texttt{NTG}(gl)$ to `true`.

2.8 `:apply`$(id_1\ id_2 \cdots id_n)$

Let *tg* be a goal such that ($\texttt{NTG}(tg) = \texttt{true}$), and let each id_i ($i \in \{1, 2, \cdots n\}$) be the built-in command name `rd-` or `rd`, or the defined name of a `:csp`$\cdots$, `:init`$\cdots$, or `(`$\cdots$`)` command (see 2.2). Then the effect of the execution of
`:apply`$(id_1\ id_2\ \cdots\ id_n)$
on $\texttt{NTG}(\_)$, $\texttt{CTM}(\_)$, $\texttt{INA}(\_)$, $\texttt{STP}(\_)$, $\texttt{DCD}(\_)$ is defined as follows.

2.8.1 id_1 = `:csp`$\cdots$

Let id_1 be the name of a `:csp` command defined as
`:def` id_1 `= :csp`$\{eq_1\ \ eq_2\ \ \cdots\ \ eq_n\}$ $\quad (n \in \{1, 2, \cdots\})$
then
`:apply`$(id_1\ id_2\ \cdots\ id_n)$ =
`:apply`(id_1)
`:select` $tg\texttt{-1}$. :apply$(id_2\ \cdots\ id_n)$
`:select` $tg\texttt{-2}$. :apply$(id_2\ \cdots\ id_n)$
$\cdots$
`:select` $tg\texttt{-n}$. :apply$(id_2\ \cdots\ id_n)$.

2.8.2 id_1 = `:init`$\cdots$

Let id_1 be the name of a `:init` command defined as
`:def` id_1 `= :init`$\cdots$
then
`:apply`$(id_1\ id_2\ \cdots\ id_n)$ = `:apply`(id_1) `:apply`$(id_2\ \cdots\ id_n)$.

2.8.3 id_1 = `(···)`

Let id_1 be the name of a `(···)` command defined as

`:def` id_1 `= (`$ids_1\ ids_2\ \cdots\ ids_m$`)` $(m \in \{1, 2, \cdots\})$

then

`:apply(`$id_1\ id_2\ \cdots\ id_n$`)` = `:apply(`$ids_1\ ids_2\ \cdots\ ids_m\ id_2\ \cdots\ id_n$`)` .

2.8.4 id_1 = `rd-` or `rd`

Let (a) tg be the goal such that (`NTG`(tg) = `true`) and (b) id_1 be `rd-` or `rd`.
If `DCD`(tg) is `true` after executing `:apply(`id_1`)` then

`:apply(`$id_1\ id_2\ \cdots\ id_n$`)` = `:apply(`id_1`)` .

If `DCD`(tg) is not `true` after executing `:apply(`id_1`)` then

`:apply(`$id_1\ id_2\ \cdots\ id_n$`)` = `:apply(`id_1`)` `:apply(`$id_2\ \cdots\ id_n$`)` .

2.9 `:show` and `:describe`

The `:show` and `:describe` commands show the status of the current proof tree. `:show` can be abbreviated as `:sh`, and `:describe` can be abbreviated as `:desc`.

▷ `:show proof`
shows all the names of goals in the current proof tree with `NTG` (_) and `DCD`(_).

▷ `:show unproved`
shows `CTM`(_), `INA`(_), `STP`(_), `DCD`(_) for all the leaf goals in the current proof tree such that (`DCD`(_) = `false`).

▷ `:show discharged`
shows the discharged sentence with its context module.

▷ `:show goal`
shows `CTM`(gl), `INA`(gl), `STP`(gl), `DCD`(gl) for the goal gl such that (`NTG`(gl) = `true`).

▷ `:show goal` gl
shows `CTM`(gl), `INA`(gl), `STP`(gl), `DCD`(gl) for the goal gl.

▷ `:show def`
shows all the command names defined with `:def` commands.

▷ `:describe proof`
shows `CTM`(_), `INA`(_), `STP`(_), `DCD`(_) for all the goals in the current proof tree.

2.10 `:red`

▷ `:red` *trm* `.`
shows the normal form of the term *trm* in the goal *tg* such that (`NTG`(*tg*) = `true`).

▷ `:red in` *gl* `:` *trm* `.`
shows the normal form of the term *trm* in the goal *gl*.

2.11 `:set`

▷ `:set(verbose,on)`
sets the PTcalc flag **verbose** on, and makes outputs from PTcalc more detailed.

▷ `:set(verbose,off)`
sets the PTcalc flag **verbose** off.

▷ `:set(verbose,show)`
shows the value of the PTcalc flag **verbose**.