Preprint of a paper to appear in the Proceedings of the 1991 International Tutorial and Workshop on the HOL Theorem Proving System, 27–30 August 1991, Davis California (IEEE Computer Society Press).

A Package for Inductive Relation Definitions in HOL

T. F. Melham

University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street
Cambridge, CB2 3QG, England.

Abstract

This paper describes a set of theorem proving tools based on a new derived principle of definition in HOL, namely the introduction of relations inductively defined by a set of rules. Such inductive definitions abound in computer science. Example application areas include reasoning about structured operational semantics, type judgements, transition relations for process algebras, reduction relations, and compositional proof systems. The package described in this paper automates the derivation of certain inductive definitions involved in these applications and provides the basic tools needed for reasoning about the relations introduced by them.

1 Introduction

The HOL user community has a strong tradition of taking a purely *definitional* approach to using higher order logic. That is, the syntax of the logic is extended with new notation not simply by postulating axioms to give meaning to it, but rather by defining it in terms of existing expressions of the logic that already have the required semantics. The advantage of this approach, as opposed to the axiomatic method, is that each of the primitive rules of definition in the HOL logic—namely, constant definition, constant specification, and type definition—is guaranteed to preserve consistency. The disadvantage is that these rules admit only definitions that satisfy certain very restrictive rules of formation. Definitions expressed in any other form must always be justified formally by deriving them from equivalent, but possibly rather complex, primitive definitions.

The ML metalanguage allows users to implement derived inference rules in the HOL system and thus provides a facility for automating proofs that justify derived rules of definition. For example, recursive definitions are not admitted by the primitive rules of definition of the HOL logic. But certain recursive type definitions and function definitions are supported in the system by derived inference rules written in ML [1, 2]. The details of the primitive definitions that underlie these rules are hidden from the user, and their ML implementations are highly optimized. So these derived principles of definition may simply be regarded as primitive by most users of the system.

This paper describes a set of theorem-proving tools based on a new derived principle of definition in HOL for defining relations inductively by a set of rules. Sections 2 and 3 give a general introduction to the class of inductive definitions handled by the package and explain the logical basis for these definitions. The remaining sections describe the ML functions provided by the package and briefly mention some applications for which the package can be used.

2 Inductive definitions

The following is a simple but typical example of a relation defined inductively by a set of rules. (This example is taken from [3].) Let $R \subseteq A \times A$ be a binary relation on a set A. The reflexive-transitive closure of R can be defined to be the least relation $R^* \subseteq A \times A$ for which the following deduction rules hold.

R1 $$\frac{}{R^*(x,y)} \quad R(x,y)$$

R2 $$\frac{}{R^*(x,x)}$$

R3 $$\frac{R^*(x,z) \qquad R^*(z,y)}{R^*(x,y)}$$

These rules state precisely the properties required of the reflexive-transitive closure of the relation R. Rule **R1** states that it must contain at least all the values in R; rule **R2** states that it must be reflexive; and rule **R3** states that it must be transitive. The reflexive-transitive closure R^* may therefore simply be *defined* to be the least relation that satisfies these conditions. It then follows simply by definition that the rules **R1**, **R2** and **R3** are in fact satisfied by R^*. Moreover, it follows immediately that R^* is a subset of any other relation that satisfies these rules, since R^* is defined to be the *least* such relation. This means that R^* contains only those pairs of values that it must contain by virtue of satisfying the rules. As will be discussed below, this property gives rise to an induction principle for reasoning about the relation R^*.

The definition given above is valid because the rules **R1**, **R2**, and **R3** make only positive statements about the elements of R^*. This guarantees that the least relation satisfying these rules does exist. In particular, if the rules have this form, then one can show that the intersection of any set of relations that satisfy the rules also satisfies the rules. Moreover, at least one binary relation satisfies the rules, namely the maximal relation $A \times A$. The 'least' or smallest relation that satisfies the rules may therefore legitimately be defined to be the intersection of all such relations.

In general, an inductive definition of an n-place relation R consists of a set of rules of the form:

$$\frac{R(t_1^1,\ldots,t_n^1) \quad \cdots \quad R(t_1^i,\ldots,t_n^i)}{R(t_1,\ldots,t_n)} \quad C_1 \cdots C_j$$

The terms above the line are the *premisses* of the rule, each of which makes a positive assertion of membership in the relation R. The term below the line, called the *conclusion* of the rule, likewise asserts membership in R. The terms C_1,...,C_j are *side conditions* on the rule; these may be arbitrary propositions not involving the relation R being defined. A relation R is *closed* under such a rule if whenever the premisses and side conditions hold, the conclusion also holds. The relation *inductively defined* by a collection of such rules is the least relation closed under all the rules.

2.1 Rule induction

By virtue of its definition as the *least* relation closed under a set of rules, every inductively defined relation comes with an associated induction principle. This principle of *rule induction* is essential for many proofs involving such relations. (The term 'rule induction' was coined by Glynn Winskel in [4]).

The principle of rule induction for an inductively defined relation may be stated briefly as follows. Let R be an n-place relation inductively defined by a set of rules, and suppose we wish to show that every element in R has a certain property P:

$$\text{if } R(x_1,\ldots,x_n) \text{ then } P[x_1,\ldots,x_n] \qquad (1)$$

Since R is the least relation closed under the rules, any relation S which is also closed under the rules has the property that $R \subseteq S$. Now, let

$$S = \{(x_1,\ldots,x_n) \mid P[x_1,\ldots,x_n]\}$$

Then to prove the desired property of R, it suffices to show that the relation S is closed under the rules that define R. For if the relation S in fact is closed under the rules, then we have that $R \subseteq S$ and therefore that every element of R has the defining property of S—i.e. statement (1) holds of the relation R.

For the relation R^* defined above, the principle of rule induction is stated as follows. In order to prove that a property $P[x, y]$ holds for all x and y for which $R^*(x, y)$, it suffices to show that:

- for all x and y, $R(x, y)$ implies $P[x, y]$
- for all x, $P[x, x]$
- for all x, y, and z, $P[x, z]$ and $P[z, y]$ imply $P[x, y]$

This is an inductive form of argument: if the property P holds in the 'base cases' corresponding to rules **R1** and **R2**, and if P is preserved by the rule **R3** (the 'step case' of the induction), then every pair in R^* has the property P. A similar induction principle holds for every relation inductively defined by a set of rules.

3 Inductive definitions in logic

Inductive definitions are based on the concept of a relation being closed under a set of rules. Since rules are essentially implications—*if* the premisses and side conditions hold, *then* the conclusion holds—it is straightforward to express this concept in logic.

Consider, for example, the rules given above for reflexive-transitive closure. Let $R : \alpha{\to}\alpha{\to}bool$ be a fixed but arbitrary relation on α. (Here, a relation is represented by a curried function; but we shall continue to speak loosely of a pair of values x and y as being 'in' the relation R when $R\ x\ y$ holds.) The following formula then asserts that a relation $P : \alpha{\to}\alpha{\to}bool$ is closed under the rules defining the reflexive-transitive closure of R:

$$\begin{array}{l}(\forall x\ y.\ R\ x\ y \supset P\ x\ y) \wedge \\ (\forall x.\ P\ x\ x) \wedge \\ (\forall x\ y.\ (\exists z.\ P\ x\ z \wedge P\ z\ y) \supset P\ x\ y)\end{array}$$

Each rule is expressed by a quantified implication of its conclusion by the conjunction of its premisses and side conditions. A rule with no side conditions or premisses is just represented by a universally quantified assertion of its conclusion. Closure of a relation under any set of rules of the form discussed above can be expressed in logic in a similar way.

Using this method of expressing the notion of closure under a set of rules, one can define the *least* relation closed under a set of rules simply by taking the intersection of all such relations. For example, a function

$$\mathsf{Rtc} : (\alpha{\to}\alpha{\to}bool) \to (\alpha{\to}\alpha{\to}bool)$$

that maps an arbitrary relation $R : \alpha{\to}\alpha{\to}bool$ to its reflexive-transitive closure $\mathsf{Rtc}\ R$ can be defined in the HOL logic by the constant definition:

$$\begin{array}{l}\vdash \forall R\ x\ y.\ \mathsf{Rtc}\ R\ x\ y\ = \\ \quad \forall P.\ ((\forall x\ y.\ R\ x\ y \supset P\ x\ y) \wedge \\ \qquad (\forall x.\ P\ x\ x) \wedge \\ \qquad (\forall x\ y.\ (\exists z.\ P\ x\ z \wedge P\ z\ y) \supset P\ x\ y)) \\ \qquad\quad \supset \\ \qquad P\ x\ y\end{array}$$

This definition states that a pair x and y is in the relation $\mathsf{Rtc}\ R$ exactly when it is in every relation P closed under the rules for reflexive-transitive closure. That is, $\mathsf{Rtc}\ R$ is defined to be the intersection of all relations closed under these rules. As will be discussed in the section that follows, this indeed makes $\mathsf{Rtc}\ R$ the least such relation, as required.

3.1 Deriving the rules and rule induction

Any relation intended to be defined inductively by a set of rules can be defined formally in the HOL logic by a constant definition of the kind illustrated by the Rtc example given above. Such a definition, however, merely introduces the relation as the intersection of all relations that satisfy the desired set of rules. The proof obligations of a derived principle of inductive definition are, first of all, to show that the resulting relation in fact does satisfy these rules, and secondly to show that it is indeed the least such relation. It is these proof obligations which are automated by the HOL inference rule described below in section 4.

In the case of the simple reflexive-transitive closure example, the first proof obligation is to show that:

$$\vdash \forall R\ x\ y.\ R\ x\ y \supset \mathsf{Rtc}\ R\ x\ y$$

$$\vdash \forall R\ x.\ \mathsf{Rtc}\ R\ x\ x$$

$$\vdash \forall R\ x\ y.\ (\exists z.\ \mathsf{Rtc}\ R\ x\ z \wedge \mathsf{Rtc}\ R\ z\ y) \supset \mathsf{Rtc}\ R\ x\ y$$

That is, one must prove that the rules **R1**, **R2**, and **R3** follow from the somewhat indirect formal definition of the relation Rtc R given in the previous section. The second proof obligation is to show that Rtc R is the least relation that satisfies these rules:

$$\begin{array}{l}\vdash \forall R\ P.\ ((\forall x\ y.\ R\ x\ y \supset P\ x\ y) \wedge \\ \qquad (\forall x.\ P\ x\ x) \wedge \\ \qquad (\forall x\ y.\ (\exists z.\ P\ x\ z \wedge P\ z\ y) \supset P\ x\ y)) \\ \qquad\quad \supset \\ \qquad \forall x\ y.\ \mathsf{Rtc}\ R\ x\ y \supset P\ x\ y\end{array}$$

This is the principle of rule induction for Rtc R. These four theorems constitute a complete statement of the defining properties of reflexive-transitive closure. All four can be proved fully automatically in HOL by the derived inference rule described in the next section.

4 Automation

The main component of the inductive definitions package is an ML function that takes as an argument a list of rules and automatically proves the defining properties of the relation inductively defined by them. More precisely, this derived HOL inference rule builds a term that denotes the least relation closed under the rules using the intersection construction described in the previous section. A constant is then introduced (via a constant specification) to name this relation. The result is a set of theorems stating that the newly-defined relation is the least relation closed under the rules supplied by the user.

The ML function that implements this principle of inductive definition is:

```
new_inductive_definition
  : bool ->                          (infix flag)
    string ->                        (defn. name)
    (term # term list) ->            (pattern)
    (term list # term) list ->       (rules)
    (thm list # thm)                 (result)
```

The first argument to this function is a boolean flag which indicates if the constant that is defined is to have infix syntactic status. The second argument is the name under which the resulting definition will be saved on disk. The third argument is a 'pattern' that supplies information which is needed because this ML function can be used to define classes of inductively defined relations, rather than just single instances of these relations. Details of the purpose and format of this pattern will be explained later. The final argument is a list of rules, each of which is represented by a pair of the form:

`([`*premisses and side conditions*`],` *conclusion*`)`

The first component is a list of the premisses and side conditions, which may be arranged in any order. The second component is the conclusion of the rule. Side conditions can be arbitrary boolean terms, provided they do not mention the relation being defined. The premisses and conclusion must be positive assertions of membership in the relation being defined. The precise form that these assertions must take is explained later, but roughly speaking the premisses and conclusion of a rule must be terms of form "R t_1 ... t_n", where

$$R : \sigma_1 \texttt{ -> } \ldots \texttt{ -> } \sigma_n \texttt{ -> bool}$$

is a variable representing the n-place relation that is to be defined, and each t_i`:`σ_i is an arbitrary term not containing R.

Given an infix flag, a name, a pattern, and a list of rules, the ML function `new_inductive_definition` automatically proves the existence of the least relation that satisfies these rules. A constant is then introduced to denote this relation using a constant specification, the result of which is saved on disk under the supplied name. The value returned is a pair consisting of a list of theorems which state that

the newly-defined relation satisfies the rules, together with a theorem asserting rule induction for the relation. These theorems give a complete statement of the defining properties for the least relation closed under the specified set of rules.

4.1 A simple example

The following example HOL session shows how the function `new_inductive_definition` can be used to inductively define the set of even natural numbers.

1

```
#let (rules,ind) =
   let Even = "Even:num->bool" in
   new_inductive_definition false ‘Even‘
   ("^Even n", [])

   [ [
     % ------------------------------ % ],
                 "^Even 0"            ;

     [           "^Even n"
     % ------------------------------ % ],
               "^Even (n+2)"        ];;
```

The first rule in this definition states that `0` is an even natural number, and the second rule states that if `n` is even then `n+2` is also even. (Antiquotation and ML comments are used to give a readable presentation of these rules.) Since the even natural numbers are exactly those numbers obtainable from zero by adding multiples of two, these rules inductively define '`Even n`' such that it holds precisely when `n` is even.

The value supplied for the pattern in this example is the pair `("Even n",[])`. The first component of this pair indicates that the constant to be defined, namely `Even`, is a one-place function with typical argument `n`. In general, the second component of a pattern is a non-empty list only when a *class* of relations is being defined (see below). In this example, `Even` is a single inductively-defined predicate, and the list component of the pattern is therefore empty.

When the definition shown in box 1 is evaluated, `new_inductive_definition` automatically proves the existence of the least predicate closed under the given list of rules and then defines the constant `Even` to denote this predicate. The following automatically-proved theorems about `Even` are then returned:

2

```
rules =
[⊢ Even 0;
 ⊢ ∀n. Even n ⊃ Even(n + 2)] : thm list
ind =
⊢ ∀P. P 0 ∧ (∀n. P n ⊃ P(n + 2)) ⊃
      (∀n. Even n ⊃ P n)
```

The theorems bound to the ML identifier `rules` state that the required rules hold of the predicate `Even`. And the rule induction theorem bound to `ind` states that the set of numbers for which `Even` holds is the least set that satisfies these rules.

An analogous set of defining theorems can be proved automatically for any particular relation inductively defined by a list of rules. The next section shows how this derived principle of inductive definition in HOL can also be used to define a parameterized class of relations.

4.2 Defining a class of relations

The constant Rtc defined in section 3 is not itself an inductively-defined relation, but rather a function that maps an arbitrary relation R to an inductively-defined relation Rtc R. The function Rtc therefore represents an entire class of inductively-defined relations, one for each possible value of R.

The information that is required by the derived rule `new_inductive_definition` in order to handle the definition of such functions is supplied by its pattern argument. In the general case, a pattern is a pair of the following form:

("$R\ v_1\ \ldots\ v_n$", ["v_i";...;"v_j"])

The first component of the pattern is an application of the n-place curried function that is to be defined (in this case, R) to n distinct variables $v_1, \ldots, v_n$. The second component is a list of those variables that occur at the positions in this application which

correspond to the parameters of class of inductively-defined relations, rather than to the actual arguments to these relations.

An example of the role of the pattern argument in defining a class of relations is provided by the following definition of reflexive-transitive closure in HOL.

3

```
#let (rules,ind) =
   let Rtc = "Rtc:(*->*->bool)->*->*->bool"
 in
   new_inductive_definition false ‘Rtc‘
   ("^Rtc R x y", ["R:*->*->bool"])

   [ [      "R (x:*) (y:*):bool"
     % ------------------------------ % ],
               "^Rtc R x y"          ;

     [
     %------------------------------- % ],
               "^Rtc R x x"          ;

     [  "^Rtc R x z";  "^Rtc R z y"
     %------------------------------- % ],
               "^Rtc R x y"          ];;
```

The pattern in this case is the pair:

```
("Rtc R x y", ["R:*->*->bool"])
```

The first component of this pattern specifies that the function `Rtc` is to take three arguments in total—a relation `R`, and two values `x` and `y`. The second part of the pattern (the list containing just `R`) specifies that the relation argument `R` is to be a parameter to the class of inductively-defined relations that will be represented by `Rtc`. The remaining variables `x` and `y` are assumed to indicate the positions of actual arguments to the predicate that represents these relations.

The result of evaluating this inductive definition in HOL is the following collection of theorems:

4

```
rules =
[⊢ ∀R x y. R x y ⊃ Rtc R x y;
 ⊢ ∀R x. Rtc R x x;
 ⊢ ∀R x y. (∃z. Rtc R x z ∧ Rtc R z y)
             ⊃
           Rtc R x y] : thm list
ind =
⊢ ∀R P.
   (∀x y. R x y ⊃ P x y) ∧
   (∀x. P x x) ∧
   (∀x y. (∃z. P x z ∧ P z y) ⊃ P x y)
      ⊃
   (∀x y. Rtc R x y ⊃ P x y)
```

Here, the ML variable `rules` has been bound to a list of theorems which state the three rules that inductively define the reflexive-transitive closure of a relation. The theorem `ind` states the corresponding principle of rule induction for an inductively-defined relation `Rtc R`.

4.3 Stating premisses and conclusions

In addition to the use of the pattern argument, the `Rtc` example also illustrates a restriction on the form in which the premisses and conclusions of rules must be supplied to `new_inductive_definition`. As was mentioned above, premisses and conclusions must be positive assertions of membership of the form

"$R\ t_1\ \ldots\ t_n$"

where R is a variable that stands for the function to be defined. The restriction is that some of the terms among the arguments $t_1, \ldots, t_n$ in such an assertion must be variables—namely, the terms that occur at positions which, according to the supplied pattern, correspond to the parameters of a class of relations. In particular, the terms that occur at these positions must be the same variables given in the pattern itself.

The rules for reflexive-transitive closure shown in box 3 conform to this restriction. Here, the pattern indicates that in the typical assertion of membership `"Rtc R x y"` (i.e. the first component of the pattern), the variable `R` marks the position of a parameter to the class of relations to be defined. Every premiss and conclusion mentioned in the rules must

therefore be a term of the form "`Rtc R` t_1 t_2", where the arguments t_1 and t_2 may be arbitrary terms but the parameter `R` must be the variable given in the pattern.

5 A tactic for rule induction

The inductive definitions package in HOL includes a number of auxiliary functions that support reasoning about inductively-defined relations, in addition to the derived rule of definition itself. The most important of these is the following general tactic for goal-directed proofs by rule induction:

```
RULE_INDUCT_THEN
 : thm ->                          (induction thm)
   (thm -> tactic) ->              (premiss cont.)
   (thm -> tactic) ->              (side cond. cont.)
   tactic                          (result)
```

The first argument to this function is the rule induction theorem returned by `new_inductive_definition` for a given inductively-defined relation. Like the general structural induction tactic in HOL, the rule induction tactic is parameterized by functions that determine what is done with induction hypotheses. These may be either premisses or side conditions, and the user may wish to treat these two kinds of induction hypotheses differently. Two separate theorem continuations are therefore supplied as the second and third arguments to the function `RULE_INDUCT_THEN`.

Given the rule induction theorem for an inductively-defined n-ary relation R, the function described above returns a specialized rule induction tactic that reduces goals of the form:

$$\text{"}\forall x_1\ \ldots\ x_n.\ R\ x_1\ \ldots\ x_n \supset P[x_1, \ldots, x_n]\text{"}$$

to the subgoal(s) of proving that the property P is preserved by the rules that inductively define R. The rule induction theorem for `Rtc`, for example, is:

5

```
#ind;;
⊢ ∀R P.
   (∀x y. R x y ⊃ P x y) ∧
   (∀x. P x x) ∧
   (∀x y. (∃z. P x z ∧ P z y) ⊃ P x y)
      ⊃
   (∀x y. Rtc R x y ⊃ P x y)
```

A rule induction tactic for `Rtc` can be constructed from this theorem by making the simple ML definition:

6

```
#let Rtc_INDUCT_TAC =
   RULE_INDUCT_THEN ind
      ASSUME_TAC ASSUME_TAC;;
Rtc_INDUCT_TAC = - : tactic
```

The use of `ASSUME_TAC` in this definition means that the induction hypotheses arising from the premisses and side conditions of the rules are to be added to the assumptions of the subgoals that are generated. The resulting rule induction tactic for `Rtc` is described by:

$$\frac{\Gamma\ \texttt{?-}\ \forall x\ y.\ \texttt{Rtc}\ R\ x\ y \supset P[x,y]}{\begin{array}{c}\Gamma \cup \{R\ x\ y\}\ \texttt{?-}\ P[x,y] \\ \Gamma\ \texttt{?-}\ \forall x.\ P[x,x] \\ \Gamma \cup \{P[x,z],\ P[z,y]\}\ \texttt{?-}\ P[x,y]\end{array}}$$

This tactic implements the induction scheme described above in section 2.1. It reduces the goal of proving that a property $P[x, y]$ holds for all pairs x and y related by `Rtc` R to showing that this property is preserved by the rules that inductively define this relation.

5.1 An example

The following session shows how the rule induction tactic for `Rtc` constructed in the previous section can be used to prove a simple theorem about this relation. The aim is to show that the reflexive-transitive closure of a symmetric relation is also symmetric. The proof begins by using the HOL subgoal package (see [1]) to set up an appropriate goal to be proved:

```
#set_goal                                                    7
   (["∀x:*. ∀y. R x y ⊃ R y x"],
     "∀x:*. ∀y. Rtc R x y ⊃ Rtc R y x");;
"∀x y. Rtc R x y ⊃ Rtc R y x"
    [ "∀x y. R x y ⊃ R y x" ]

() : void
```

The assumption of the goal is that the relation `R` is symmetric, and the conclusion states that the closure `Rtc R` is also symmetric. The conclusion of the goal is in precisely the right form for a proof by rule induction using the induction tactic described above. Applying this tactic results in:

```
#expand Rtc_INDUCT_TAC;;                                     8
OK..
3 subgoals
"Rtc R y x"                                    (subgoal 1)
    [ "∀x y. R x y ⊃ R y x" ]
    [ "Rtc R z x" ]
    [ "Rtc R y z" ]

"∀x. Rtc R x x"                                (subgoal 2)
    [ "∀x y. R x y ⊃ R y x" ]

"Rtc R y x"                                    (subgoal 3)
    [ "∀x y. R x y ⊃ R y x" ]
    [ "R x y" ]

() : void
```

Subgoals 1 and 2 are trivial, since the relation `Rtc R` is transitive and reflexive by definition. The tactic proofs for these subgoals can simply use the rules shown above in box 4. The proof of subgoal 3 is also easy. The proposition `"R y x"` follows immediately from the two assumptions of the subgoal; and this proposition together with the fact that

$$\vdash \forall \texttt{R x y. R x y} \supset \texttt{Rtc R x y}$$

directly entail the required conclusion.

The proof sketched above is a trivial example of the kind of reasoning sometimes referred to as induction over the structure (or the depth) of derivations in a deductive system stated by a set of rules. This form of inductive argument, which is very common in certain areas of theory (for example, operational semantics and process algebras), is made directly accessible in HOL by the tactic described in this section.

6 Tactics and inference rules

In addition to the rule induction tactic described above, the inductive definitions package also provides mechanized support for generating tactics from the theorems that state the rules for an inductively-defined relation. This takes the form of an ML function:

```
RULE_TAC : thm -> tactic
```

The theorem argument to this function is expected to be a rule expressed in the form proved by the derived principle of inductive definition described in section 4. Given such a theorem, `RULE_TAC` constructs a tactic that inverts the rule stated by it. The resulting tactic reduces goals that match the conclusion of the rule to subgoals consisting of the corresponding instances of its premisses and side conditions.

Consider, for example, the theorem which states the transitivity rule for `Rtc`:

$$\vdash \forall \texttt{R x y. (}\exists \texttt{z. Rtc R x z} \wedge \texttt{Rtc R z y)} \supset \texttt{Rtc R x y}$$

When applied to this theorem, the function `RULE_TAC` returns the tactic described by:

$$\frac{\Gamma \texttt{ ?- Rtc } R\ x\ y}{\Gamma \texttt{ ?- } \exists z.\ \texttt{Rtc } R\ x\ z \wedge \texttt{Rtc } R\ z\ y}$$

This tactic can then be used in goal-directed proofs about membership in the inductively-defined relation `Rtc` R. The other two rules that define `Rtc` R can also be converted into tactics using the function

RULE_TAC. The result is a complete set of HOL tactics for goal-directed proofs in the deductive system comprising the three rules that define reflexive-transitive closure.

It is intended that the inductive definitions package will also include a function that maps rules stated as theorems to forward inference rules in HOL (i.e. to ML functions). For example, the transitivity theorem shown above can be used to implement the following derived inference rule:

$$\frac{\Gamma_1 \vdash \texttt{Rtc}\ R\ x\ z \qquad \Gamma_2 \vdash \texttt{Rtc}\ R\ z\ y}{\Gamma_1 \cup \Gamma_2 \vdash \texttt{Rtc}\ R\ x\ y}$$

Any rule expressed as a theorem of the form proved by the derived principle of inductive definitions can likewise be converted into a forward inference rule. A function that automatically constructs such rules has not yet been implemented, partly because it has not been found necessary for the applications done so far (see section 8). For completeness, however, the author intends in future to add this function to the inductive definitions package.

7 Case analysis

The final major component of the HOL package for inductive definitions is an ML function that proves an exhaustive case analysis theorem for any given relation inductively defined by a set of rules. The name and type of this function are:

```
derive_cases_thm : (thm list # thm) -> thm
```

The arguments to this function are the list of rules satisfied by an inductively defined relation, together with its rule induction theorem. (These are precisely the defining theorems which are proved and returned by `new_inductive_definition`.) When supplied with these theorems, `derive_cases_thm` proves that if an assertion of membership in the relation holds, then it holds only by virtue of the fact that one of the rules can be used to derive it. This allows one to drive the rules that define a relation 'backwards', inferring from the conclusion of one of the rules that the premisses and side conditions hold.

The following interaction with the HOL system shows the theorem proved by `derive_cases_thm` for the `Rtc` example introduced above. The ML variables `rules` and `ind` are assumed to have the bindings shown above in box 4.

9

```
#derive_cases_thm (rules,ind);;
⊢ ∀R x y.
    Rtc R x y ⊃
      R x y   ∨
      (y = x) ∨
      (∃z. Rtc R x z ∧ Rtc R z y)
```

Roughly speaking, the resulting theorem states that if `Rtc R x y` holds, then either:

- it is derivable by the inclusion rule **R1**, in which case `x` and `y` are related by `R`; or
- it is derivable by the reflexivity rule **R2**, in which case `x` and `y` are equal; or
- it is derivable by the transitivity rule **R3**, in which case there must be an intermediate value `z` such that `Rtc R x z` and `Rtc R z y`.

A similar theorem can be proved automatically for any relation defined inductively using the package. Work is currently underway to strengthen this theorem from an implication to an equation, so that it can be used for rewriting.

8 Applications

In a joint project with Juanito Camilleri, a set of example proofs has been developed to illustrate the potential for applications of the inductive definitions package. These examples include: the definition of an operational semantics for a simple programming language and a proof that its evaluation relation is deterministic; the definition of a reduction relation for combinatory logic and a proof that it has the Church-Rosser property; a definition of provability in a Hilbert style proof system for minimal intuitionistic logic; the definition of a type system for combinatory

logic and a proof of the Curry-Howard isomorphism for typed combinatory logic and minimal intuitionistic logic; and definitions of the trace and transition semantics for a simple process algebra, together with the proof of a formal statement of the relationship between them. A report on this work is in preparation, and the HOL source code for the examples will be made available to interested users.

References

[1] DSTO, The University of Cambridge, and SRI International, *The HOL System: DESCRIPTION* (1991).

[2] T. F. Melham, 'Automating Recursive Type Definitions in Higher Order Logic', in: *Current Trends in Hardware Verification and Automated Theorem Proving*, edited by G. Birtwistle and P.A. Subrahmanyam (Springer-Verlag, 1989), pp. 341–386.

[3] A. M. Pitts, 'Semantics of Programming Languages', unpublished lecture notes, University of Cambridge Computer Laboratory (October 1989).

[4] G. Winskel, 'Introduction to the Formal Semantics of Programming Languages', unpublished lecture notes, University of Cambridge Computer Laboratory (October 1985).