PROGRAM VERIFICATION II: COMPARING CONDITIONAL RULES

CS 340, FALL 2004

PURPOSE

This document discusses conditional rules and disjoint rules, and how to convert conditional rules to disjoint rules, and how to compare two conditional rules. A long example is given in which a program is compared to a specification.

CONTENTS

Purpose 1
1. Conditional Rules 1
2. Disjoint Rules 2
3. Comparing Conditional Rules 3
4. A Longer Example 4
4.1. Converting Conditional Rules to Disjoint Rules 4
4.2. Comparing Disjoint Rules: Domains 6
4.3. Comparing Disjoint Rules: Mappings 6
4.4. Conclusion 8
4.5. Comparing Domains: A Simpler Method 8

1. CONDITIONAL RULES

Conditional rules are essentially domain restrictions of other functions. For example, a function expressed by concurrent assignment is $x,y := x+y,x-y$. This concurrent assignment expresses a function which maps one state to a new state. For example, this concurrent assignment imposes the following mapping:

$$\{\langle x,3\rangle,\langle y,5\rangle\} \mapsto \{\langle x,8\rangle,\langle y,-2\rangle\}.$$

Another such function is $x,y := x+y,y-x$. Now, by combining these two functions on restricted domains, we can create a new function expressed as a conditional rule:

$$\begin{array}{lcl} (x>y & \Longrightarrow & x,y := x+y,x-y \\ |\,x\leq y & \Longrightarrow & x,y := x+y,y-x). \end{array}$$

(This new conditional rule expresses the function $x,y := x+y,|x-y|$.) The vertical bar in the conditional rule can be read as "else" or "otherwise." In this case, the predicate for the second part of the rule could be dropped, giving the simpler form $(x>y \implies x,y := x+y,x-y\,|\,x,y := x+y,y-x)$. This is effectively the same as $(x>y \implies x,y := x+y,x-y\,|\,T \implies x,y := x+y,y-x)$.

Date: 7th October 2004.

Conditional rules can be nested, and have as many conditions as necessary. Consider the following conditional rule:

$$(p \implies (q_{1,1} \implies r_{1,1} \,|\, q_{1,2} \implies r_{1,2} \,|\, r_{1,3}) \,|\, r_2).$$

This conditional rule can be directly expressed in the C language as follows:

```
if (p) {
    if (q1,1) {
        r1,1;
    } else if (q1,2) {
        r1,2;
    } else {
        r1,3;
    }
} else {
    r2;
}
```

Conditional rules do not necessarily have nice algebraic properties. For example, they cannot be re-ordered. It should be obvious that the rule $(p_1 \implies r_1 \,|\, p_2 \implies r_2)$ is not necessarily the same as $(p_2 \implies r_2 \,|\, p_1 \implies r_1)$. Assume both predicates p_1 and p_2 can be true at the same time. Then the first rule gives outcome r_1, since p_1 is evaluated first, whereas the second rule gives outcome r_2, since p_2 is evaluated first.

Example 1.1. Consider the following C source code.

```
if (x > 5) {
    y = y + 1;
} else if (x > 0) {
    y = y - x;
}
```

This code executes the body of the second if statement when the value of x is in the interval (0,5]. Naïvely re-ordering the rules results in the following code.

```
if (x > 0) {
    y = y - x;
} else if (x > 5) {
    y = y + 1;
}
```

This code fragment *never* executes the body of the second if statement.

Consider the following conditional rule:

$$\begin{array}{l}(p_1 \implies (q_{1,1} \implies r_{1,1} \,|\, q_{1,2} \implies r_{1,2}) \\ |\, p_2 \implies (q_{2,2} \implies r_{2,1} \,|\, q_{2,2} \implies r_{2,2})).\end{array}$$

If we push the outer predicates inside (distribute), we get the following conditional rule:

$$\begin{array}{l}(p_1 \wedge q_{1,1} \implies r_{1,1} \\ |\, p_1 \wedge q_{1,2} \implies r_{1,2} \\ |\, p_2 \wedge q_{2,1} \implies r_{2,1} \\ |\, p_2 \wedge q_{2,2} \implies r_{2,2}).\end{array}$$

Are these two conditional rules equivalent? Consider $p_1 = T$, $q_{1,1} = q_{1,2} = F$, $q_{2,1} = T$, and $q_{2,2} = F$. The first conditional rule *is undefined* in this case. The second conditional rule

computes $r_{2,1}$. Conditions cannot be distributed or factored like this. Since if statements in programming languages are conditional rules, this is a source of trouble when programmers try to "simplify" conditional logic in programs.

2. Disjoint Rules

Consider the following conditional rule:

$$(p \implies r_1 \,|\, \neg p \implies r_2).$$

This conditional rule *can* be re-ordered, giving the equivalent conditional rule:

$$(\neg p \implies r_2 \,|\, p \implies r_1).$$

The reason for this is that the conditions are *disjoint*. Conditional rules, all of whose conditions are disjoint, are *disjoint rules*. Disjoint rules can not only be re-ordered, but they can be distributed and factored, unlike conditional rules.

To create a disjoint rule from a conditional rule, "and" each predicate with the negation of all preceding predicates. Consider the following conditional rule:

$$(p_1 \implies r_1 \,|\, p_2 \implies r_2 \,|\, p_3 \implies r_3).$$

For the first predicate, nothing needs to be done. For the second predicate, it must be composed with $\neg p_1$ to obtain $\neg p_1 \wedge p_2$. The third predicate must be composed with $\neg p_1 \wedge \neg p_2$ to obtain $\neg p_1 \wedge \neg p_2 \wedge p_3$. The final disjoint rules is as follows:

$$(p_1 \implies r_1 \,|\, \neg p_1 \wedge p_2 \implies r_2 \,|\, \neg p_1 \wedge \neg p_2 \wedge p_3 \implies r_3).$$

Note that this does not change the meaning of the rule; it simply makes the "else" condition explicit.

3. Comparing Conditional Rules

Given two conditional rules which are not obviously equivalent, they can be compared by applying the following procedure.

(1) Convert both rules to disjoint rules.
(2) Determine the effective domain of each conditional rule (the set of values on which the rule produces some action). This can be done by combining all the rule's predicates with logical "or."
(3) Compare the effective domains. If the rules do not have the same domain, they are not the same. One domain might contain the other, or the domains might overlap. In these cases, one rule might be *sufficiently* correct, or *partially* correct.[1]
(4) Construct all pairs of predicates, in which one predicate is taken from the first rule, and the other is taken from the second rule, and "and" the predicates. Compare the results implied by the two rules for equality.

Example 3.1. Consider the following conditional rules:

$$\begin{aligned} C_1 &= (p_1 \implies r_1 \,|\, p_2 \implies r_2) \\ C_2 &= (q_1 \implies s_1 \,|\, q_2 \implies s_2 \,|\, s_3). \end{aligned}$$

[1] Consider these rules: $(x = 0 \implies x := 2x)$ and $(x \geq 0 \implies x := 0)$. The effective domains are not the same ($x = 0$ versus $x > 0$), but they overlap when $x = 0$. The second rule might be said to be *sufficiently correct* with respect to the first.

These rules are converted to the following disjoint rules:

$$\begin{aligned} D_1 &= (p_1 \implies r_1 \,|\, \neg p_1 \wedge p_2 \implies r_2) \\ D_2 &= (q_1 \implies s_1 \,|\, \neg q_1 \wedge q_2 \implies s_2 \,|\, \neg q_1 \wedge \neg q_2 \implies s_3). \end{aligned}$$

The next step is to show that the domains are the same. For complete correctness one must show the following:

$$p_1 \vee (\neg p_1 \wedge p_2) \iff q_1 \vee (\neg q_1 \wedge q_2) \vee (\neg q_1 \wedge \neg q_2).$$

Alternately, to show that D_2 is sufficiently correct with respect to D_1, one must show the following:

$$p_1 \vee (\neg p_1 \wedge p_2) \implies q_1 \vee (\neg q_1 \wedge q_2) \vee (\neg q_1 \wedge \neg q_2).$$

That is, the domain of D_2 contains the domain of D_1.

After the domain test, it is necessary to show that the two rules agree on all pairwise conjunctions. The first rule has two predicates, and the second rule has three, so there are six pairs. One must prove all the following assertions:

(1) $p_1 \wedge q_1 \implies r_1 = s_1$
(2) $p_1 \wedge \neg q_1 \wedge q_2 \implies r_1 = s_2$
(3) $p_1 \wedge \neg q_1 \wedge \neg q_2 \implies r_1 = s_3$
(4) $\neg p_1 \wedge p_2 \wedge q_1 \implies r_2 = s_1$
(5) $\neg p_1 \wedge p_2 \wedge \neg q_1 \wedge q_2 \implies r_2 = s_2$
(6) $\neg p_1 \wedge p_2 \wedge \neg q_1 \wedge \neg q_2 \implies r_2 = s_3$

Note that each assertion is an implication. It will be true if the premise is false, or if the conclusion is true given the premise.

---

4. A Longer Example

Let f be the specification of the required behavior of a program, defined by the conditional rule:

$$f = (x \geq 0 \wedge y \geq 0 \implies x,y := x-y,y \,|\, x \geq 0 \wedge x+y \geq 0 \implies x,y := x+y,y).$$

The following program P is presented:

```
if (x+y>=0 && y<=0) {
    x = x+y;
} else if (x+y>0 && x>=0) {
    x = x-y;
}
```

Verification of the program requires that one prove $f = [P]$ for complete correctness, or $f \subseteq [P]$ for sufficient correctness. In this case, neither is obvious. The specification and the program have different predicates.

4.1. Converting Conditional Rules to Disjoint Rules.

The program function can be written as a conditional rule:

$$\begin{aligned} [P] = (x+y \geq 0 \wedge y \leq 0 &\implies x,y := x+y,y \\ \,|\, x+y > 0 \wedge x \geq 0 &\implies x,y := x-y,y \,|\, x,y := x,y). \end{aligned}$$

Note the identity at the end. If neither of the predicates in the program is true, then the values of the variables remain unchanged; the program computes the identity. This can be seen by noting that the last `else` clause is omitted. *Be sure* to watch for the implicit identity function in if-then structures.

The two conditional rules can be re-written as equivalent disjoint rules. To do this, one "ands" each predicate with the negation of the previous predicates. This explicitly captures the "else" part of the conditional rule.

The first predicate of f is $x \geq 0 \wedge y \geq 0$. Negating this and applying DeMorgan's rule gives:

$$\begin{aligned} \neg(x \geq 0 \wedge y \geq 0) &\iff \neg(x \geq 0) \vee \neg(y \geq 0) \\ &\iff x < 0 \vee y < 0. \end{aligned}$$

This must be "anded" with the second predicate. Using distribution and basic identities, the new predicate is:

$$\begin{aligned} &(x < 0 \vee y < 0) \wedge x \geq 0 \wedge x + y \geq 0 \\ &\quad\iff ((x < 0 \wedge x \geq 0) \vee (y < 0 \wedge x \geq 0)) \wedge x + y \geq 0 \\ &\quad\iff (F \wedge (y < 0 \wedge x \geq 0)) \wedge x + y \geq 0 \\ &\quad\iff y < 0 \wedge x \geq 0 \wedge x + y \geq 0 \\ &\quad\iff y < 0 \wedge x + y \geq 0. \end{aligned}$$

The last step is due to the fact that the weakest term of an "and" can be dropped. That is, if $p \implies q$, then $p \wedge q$ can be reduced to just p (since q is implied).[2] In this case, $y < 0 \wedge x + y \geq 0 \implies x \geq 0$. Thus the new, equivalent disjoint rule is:

$$\begin{aligned} f \quad = \quad & (x \geq 0 \wedge y \geq 0 \implies x, y := x - y, y \\ & \mid y < 0 \wedge x + y \geq 0 \implies x, y := x + y, y). \end{aligned}$$

Next the program function's conditional rule must be converted into an equivalent disjoint rule. The first predicate is $x + y \geq 0 \wedge y \leq 0$. Negating this gives:

$$\begin{aligned} \neg(x + y \geq 0 \wedge y \leq 0) &\iff \neg(x + y \geq 0) \vee \neg(y \leq 0) \\ &\iff x + y < 0 \vee y > 0. \end{aligned}$$

Again, this must be "anded" with the second predicate to obtain the new predicate. This gives:

$$\begin{aligned} &(x + y < 0 \vee y > 0) \wedge x + y > 0 \wedge x \geq 0 \\ &\quad\iff ((x + y < 0 \wedge x + y > 0) \vee (y > 0 \wedge x + y > 0)) \wedge x \geq 0 \\ &\quad\iff (F \vee (y > 0 \wedge x + y > 0)) \wedge x \geq 0 \\ &\quad\iff y > 0 \wedge x + y > 0 \wedge x \geq 0 \\ &\quad\iff y > 0 \wedge x \geq 0. \end{aligned}$$

Note that since $y > 0 \wedge x \geq 0 \implies x + y > 0$, the weak term $x + y > 0$ is dropped in the last step. The last clause of the disjoint rule requires the negation of both prior predicates. The first has already been negated. Negating the second one gives:

$$\begin{aligned} \neg(x + y > 0 \wedge x \geq 0) &\iff \neg(x + y > 0) \vee \neg(x \geq 0) \\ &\iff x + y \leq 0 \vee x < 0. \end{aligned}$$

[2] It might help to think of this in the analogous world of set theory. Here $p \implies q$ can be replaced with $P \subseteq Q$, and $p \wedge q$ can be replaced with $P \cap Q$. For example, let $P = \{x | p(x)\}$ and let $Q = \{x | q(x)\}$. Given that $P \subseteq Q$, it is obvious that $P \cap Q = P$. The same reasoning can be applied to replace $p \vee q$ with q when $p \implies q$.

This must be "anded" with the negation of the first predicate to get the new third predicate.

$$\begin{array}{ll} & (x+y<0\vee y>0)\wedge(x+y\leq 0\vee x<0)\\ \iff & x+y<0\vee(x+y<0\wedge x<0)\vee(y>0\wedge x+y\leq 0)\vee(y>0\wedge x<0)\\ \iff & x+y<0\vee(y>0\wedge x+y\leq 0)\vee(y>0\wedge x<0)\\ \iff & x+y<0\vee(y>0\wedge x<0)\\ \iff & x+y<0\vee x<0. \end{array}$$

The new, equivalent disjoint rule is:

4.2. **Comparing Disjoint Rules: Domains.** Disjoint rules can be re-ordered (conditional rules *cannot*). Re-organizing the two disjoint rules reveals that they are very close, but still not identical:

$$\begin{array}{rcl} f & = & (x\geq 0\wedge y\geq 0 \implies x,y := x-y,y\\ & & |\, y<0\wedge x+y\geq 0 \implies x,y := x+y,y) \end{array}$$

$$\begin{array}{rcl} [P] & = & (x\geq 0\wedge y>0 \implies x,y := x-y,y\\ & & |\, y\leq 0\wedge x+y\geq 0 \implies x,y := x+y,y\\ & & |\, x+y<0\vee x<0 \implies x,y := x,y). \end{array}$$

The two disjoint rules are identical only if they have the same domain. The domain of a disjoint rule is the "or" of all its conditions. The domain of the disjoint rule for f is:

$$\begin{array}{ll} & (x\geq 0\wedge y\geq 0)\vee(y<0\wedge x+y\geq 0)\\ \iff & (x\geq 0\wedge y\geq 0\wedge x+y\geq 0)\vee(x\geq 0\wedge y<0\wedge x+y\geq 0)\\ \iff & (x\geq 0\wedge x+y\geq 0)\wedge(y\geq 0\vee y<0)\\ \iff & (x\geq 0\wedge x+y\geq 0)\wedge T\\ \iff & (x\geq 0\wedge x+y\geq 0). \end{array}$$

In the above reduction, redundant elements are added to create common elements to be factored. For example, if $x\geq 0$ and $y\geq 0$, then it follows that $x+y\geq 0$, so this can be added to the first term. If $y<0$ and $x+y\geq 0$, then $x>0$, so $x\geq 0$ can be added to the second term. Then the term $x\geq 0\wedge x+y\geq 0$ appears in both terms.

The domain of the program function for $[P]$ is:

$$\begin{array}{ll} & (x\geq 0\wedge y>0)\vee(y\leq 0\wedge x+y\geq 0)\vee x+y<0\vee x<0\\ \iff & ((x\geq 0\vee x<0)\wedge(y>0\vee x<0))\vee(y\leq 0\wedge x+y\geq 0)\vee x+y<0\\ \iff & (T\wedge(y>0\vee x<0))\vee(y\leq 0\wedge x+y\geq 0)\vee x+y<0\\ \iff & y>0\vee x<0\vee(y\leq 0\wedge x+y\geq 0)\vee x+y<0\\ \iff & (y>0\vee x+y<0)\vee(y\leq 0\wedge x+y\geq 0)\vee x<0\\ \iff & \neg(y\leq 0\wedge x+y\geq 0)\vee(y\leq 0\wedge x+y\geq 0)\vee x<0\\ \iff & T\vee x<0\\ \iff & T \end{array}$$

In this case, the domain of the program function of $[P]$ is universal. This is because either the program fragment causes some change, or *in every other case* it computes the identity.

It is clear from the above that the two disjoint rules *do not* have the same domain, but the domain of f is a proper subset of the domain of $[P]$.[3] Thus it is still possible for P to be sufficiently correct with respect to f.

4.3. **Comparing Disjoint Rules: Mappings.** Even if the two disjoint rules have the same domain, they may result in different mappings. To demonstrate that the two rules are the same (or to discover that they are not), consider each pairwise "and" of the predicates and then show that the implied result is the same in both cases.

There are two predicates in the disjoint rule for f and three predicates in the disjoint rule for $[P]$. Thus there are six combinations to consider.

Case I. Consider the "and" of the first predicates from both disjoint rules:

$$x \geq 0 \wedge y \geq 0 \wedge x \geq 0 \wedge y > 0 \quad \Longrightarrow \quad (x,y := x-y,y) = (x,y := x-y,y).$$

Without much evaluation, this statement is clearly true, since the conclusion of the implication $(x,y := x-y,y) = (x,y := x-y,y)$ is certain. Remember that an implication $p \Longrightarrow q$ is true if either p is false or q is true.

Case II. Consider the "and" of the first predicate from f and the second predicate from $[P]$:

$$\begin{aligned} x \geq 0 \wedge y \geq 0 \wedge y \leq 0 \wedge x+y \geq 0 \quad &\Longrightarrow \quad (x,y := x-y,y) = (x,y := x+y,y) \\ x \geq 0 \wedge y = 0 \wedge x+y \geq 0 \quad &\Longrightarrow \quad (x,y := x-0,y) = (x,y := x+0,y) \\ x \geq 0 \wedge y = 0 \quad &\Longrightarrow \quad (x,y := x,y) = (x,y := x,y). \end{aligned}$$

Again, this is clearly true.

Case III. Consider the "and" of the first predicate from f and the third predicate from $[P]$:

$$\begin{aligned} x \geq 0 \wedge y \geq 0 \wedge (x+y < 0 \vee x < 0) \quad &\Longrightarrow \quad (x,y := x-y,y) = (x,y := x,y) \\ ((x \geq 0 \wedge x+y < 0) \vee (x \geq 0 \wedge x < 0)) \wedge y \geq 0 \quad &\Longrightarrow \quad (x,y := x-y,y) = (x,y := x,y) \\ ((x \geq 0 \wedge x+y < 0) \vee F) \wedge y \geq 0 \quad &\Longrightarrow \quad (x,y := x-y,y) = (x,y := x,y) \\ x \geq 0 \wedge x+y < 0 \wedge y \geq 0 \quad &\Longrightarrow \quad (x,y := x-y,y) = (x,y := x,y) \\ F \quad &\Longrightarrow \quad (x,y := x-y,y) = (x,y := x,y). \end{aligned}$$

Since an implication is true if the premise is false, this statement is true.

Case IV. Consider the "and" of the second predicate from f and the first predicate from $[P]$:

$$\begin{aligned} y < 0 \wedge x+y \geq 0 \wedge x \geq 0 \wedge y > 0 \quad &\Longrightarrow \quad (x,y := x+y,y) = (x,y := x-y,y) \\ y < 0 \wedge y > 0 \wedge x+y \geq 0 \wedge x \geq 0 \quad &\Longrightarrow \quad (x,y := x+y,y) = (x,y := x-y,y) \\ F \wedge x+y \geq 0 \wedge x \geq 0 \quad &\Longrightarrow \quad (x,y := x+y,y) = (x,y := x-y,y) \\ F \quad &\Longrightarrow \quad (x,y := x+y,y) = (x,y := x-y,y). \end{aligned}$$

Since an implication is true if the premise is false, this statement is true.

[3] Let p be the expression for the domain of f. This is the characteristic predicate of a set: $P = \{x | p(x)\}$. For the program, we have $Q = \{x | T\}$, which is the universe. Clearly $P \subseteq Q$. Equivalently, note that $p \Longrightarrow T$ is true *regardless* of the value of p.

Case V. Consider the "and" of the second predicates of f and $[P]$:

$$y < 0 \wedge x+y \geq 0 \wedge y \leq 0 \wedge x+y \geq 0 \quad \Longrightarrow \quad (x,y := x+y,y) = (x,y := x+y,y).$$

Again, the conclusion of the implication is true, so the entire implication is true independent of whether the premise is true or false.

Case VI. Consider the "and" of the second predicate of f and the third predicate of $[P]$:

$$\begin{aligned}
y < 0 \wedge x+y \geq 0 \wedge (x+y < 0 \vee x < 0) &\Longrightarrow (x,y := x+y,y) = (x,y := x,y) \\
y < 0 \wedge ((x+y \geq 0 \wedge x+y < 0) \vee (x+y \geq 0 \wedge x < 0)) &\Longrightarrow (x,y := x+y,y) = (x,y := x,y) \\
y < 0 \wedge (F \vee (x+y \geq 0 \wedge x < 0)) &\Longrightarrow (x,y := x+y,y) = (x,y := x,y) \\
y < 0 \wedge x+y \geq 0 \wedge x < 0 &\Longrightarrow (x,y := x+y,y) = (x,y := x,y) \\
F &\Longrightarrow (x,y := x+y,y) = (x,y := x,y).
\end{aligned}$$

Since an implication is true if the premise is false, this statement is true.

4.4. **Conclusion.** The program P computes the same answer as f on the shared domain. Since the domain of f is a proper subset of the domain of $[P]$, it follows that $f \subset [P]$, and P is sufficiently correct with respect to f, but not completely correct.

Could P be made completely correct? A program is going to give some result for any input combination. In this case, the best that can be done is to re-write P to generate an exception or error message when a value is provided which is outside the domain of f.

4.5. **Comparing Domains: A Simpler Method.** In this case there are two variables, x and y. These can be regarded as coordinates, and plotted on a coordinate axis. This transforms the problem into a graphical problem involving set union and intersection.

- $x \geq 0$ is the vertical axis and everything to the right of it
- $y \geq 0$ is the horizontal axis and everything above it
- $x+y \geq 0$ is the diagonal at 135° and everything above it

The "and" of two predicates takes the intersection of the two corresponding regions. The "or" of two predicates takes the union of the two regions.

Using these observations, the domain of the disjoint rules can be graphed. For example, the domain of f is as follows: