Experiments in Constrained Expression Analysis

George S. Avrunin

COINS Technical Report 87-125
November 1987

Software Development Laboratory
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

This research was partially supported by the Defense Advanced Research Projects Agency (ARPA Order No. 6104) through National Science Foundation grant CCR-8704478.

1 Introduction

This report describes some experiments in analyzing concurrent software systems using the constrained expression formalism [1,6]. The primary analysis techniques used are the inequality-based methods of [1] and [2], and the major goal of these experiments is the identification of useful heuristics for generating the necessary systems of inequalities. In particular, we are interested in determining the extent to which this process can be automated.

A couple of warnings to the reader are in order. First, this report is a description of the actual analysis of some distributed systems, and not an outline of how such an analysis might be most efficiently or elegantly conducted. We learned many things about our methods during these experiments, and this is reflected in changes in their application as the analysis progressed. Second, we assume that the reader has some familiarity with the constrained expression formalism and the design language CEDL. The material in [1] and [3] is probably sufficient background for the reader who is familiar with Ada and willing to struggle a bit with the constrained expression formalism. For fuller descriptions of constrained expressions and CEDL, the reader should also consult [6] and [4].

The systems analyzed in this report are based on the automated gas station examples given by Helmbold and Luckham in [7] to illustrate their run-time monitoring approach to debugging Ada tasking programs. These examples represent a number of iterative corrections to a system representing an automated self-service gas station, in which customers repeatedly prepay, pump gas, and collect change. In the examples, the customers, the pumps, and a central operator are all represented by Ada tasks, and a number of packages are provided to queue the customers, generate the amounts prepaid and pumped, and so forth. In the systems described by Helmbold and Luckham, there are three pump tasks and ten customer tasks.

For these experiments, we translated three of Helmbold and Luckham's systems into CEDL, an Ada-based design language we have developed for use with the constrained expression formalism. A detailed descriptions of CEDL is given in [4]. CEDL focuses on the expression of communication and synchronization among the tasks in a concurrent system, and language features not related to concurrency are kept to a minimum. Thus, for example, data types are limited, but almost all of the Ada control-flow constructs and the various forms of the Ada `select` statement have correspondents in CEDL.

In addition to translating these systems into CEDL, we reduced their size to make it practical to carry out the analysis by hand. The effects of this reduction on the behavior of the systems will be explored in later sections.

We do not present a complete analysis of the three systems here. Instead we focus on a single question. Phrased in terms of the gas station that these systems model, that question is: Does a customer who prepays always get to pump gas? In the following sections, we will show how this question can be answered by the constrained expression analysis techniques.

2 A Two-Customer System

2.1 The CEDL System

In this section, we analyze a CEDL system representing an automated self-service gas station with one pump and two customers. Our goal is to determine whether a customer who prepays can then always pump gas. The task and package declarations for this system are shown in Figure 1, and the task bodies are shown in Figures 2 and 3.

In this system, customers repeatedly arrive at the gas station and prepay for gas. This is represented by the rendezvous between a `CUSTOMER` task and the `OPERATOR` task at the `PREPAY` entry of the `OPERATOR`. If no customer is waiting, the operator activates the pump. Otherwise, the operator enters the customer's request in a queue. These activities are represented by statements in the `PREPAY` alternative of the select statement in the `OPERATOR` body.

After prepaying, a customer goes to the pump and starts it, pumps gas, and then stops the pump. These activities are represented by the two calls to entries of the `PUMP` in the body of the `CUSTOMER` task. The customer then collects change from the operator, as indicated by the accept statement in the body of the `CUSTOMER` task.

After a customer has shut it off, the pump reports to the operator. This is modelled by the call to `OPERATOR.CHARGE` in the accept `FINISH_PUMPING` statement in the body of the `PUMP` task. The operator, who waits for a customer to prepay or for a report from the pump, gives change to the customer after this report. If another customer is waiting, the operator then reactivates the pump. These activities are modelled by the statements in the accept alternative for `CHARGE` in the `OPERATOR` task body.

This system is a CEDL version of the second example studied by Helmbold and Luckham in [7]. The most significant difference between our system and theirs is that we have reduced the number of customers to two and the number of pumps to one in the CEDL system. These reductions were made to keep the constrained expressions small enough to analyze by hand. The other differences result from the focus on concurrency in CEDL, and the consequent limitations on data types and packages. Thus, we use the variables `CURRENT` and `WAITING` in the CEDL system to keep track of the customers, while Helmbold and Luckham use a queue provided in a separate package.

2.2 The Task Expressions

To begin our analysis, we must first obtain a constrained expression representation for this system. Translation rules for producing constrained expression representations from CEDL designs have been developed jointly by Laura Dillon at the University of California, Santa Barbara [3] and Usha Sundaram, Jack Wileden, and the author at the University of Massachusetts, Amherst. In Fig-

```
package COMMON is
  type C_NAME is (c1,c2);  -- names for two customers
  type COUNTER is (zero,one,two,three);
                           -- enough to handle 3 customers
end COMMON;

use COMMON;
task OPERATOR is
  entry PREPAY(CUSTOMER_ID : in C_NAME);
  entry CHARGE;
end OPERATOR;

task PUMP is
  entry ACTIVATE;
  entry START_PUMPING;
  entry FINISH_PUMPING;
end PUMP;

use COMMON;
task CUSTOMER_1 is
  entry CHANGE;
end CUSTOMER_1;

use COMMON;
task CUSTOMER_2 is
  entry CHANGE;
end CUSTOMER_2;
```

Figure 1: Task declarations for the two-customer system

```
use COMMON;
task body OPERATOR is
  CUSTOMERS : COUNTER := zero;
  CURRENT, WAITING : C_NAME;
  begin
    loop
      select
        accept PREPAY(CUSTOMER_ID : in C_NAME) do
          CUSTOMERS := COUNTER'succ(CUSTOMERS);
          if CUSTOMERS = one then     -- if no previous customer
                                      -- is waiting
            CURRENT := CUSTOMER_ID;   -- mark this one as current
            PUMP.ACTIVATE;            -- and activate the pump
           else
            WAITING := CUSTOMER_ID;    -- otherwise, mark this one
                                       -- as next in line
          end if;
        end PREPAY;
      or
        accept CHARGE;
        if CURRENT = c1 then
          CUSTOMER_1.CHANGE;
         else
          CUSTOMER_2.CHANGE;
        end if;
        CUSTOMERS := COUNTER'pred(CUSTOMERS);
        if CUSTOMERS > zero then    -- if another customer is
          CURRENT := WAITING;       -- waiting, promote that one
                                    -- to be current
          PUMP.ACTIVATE;            -- and activate the pump
        end if;
      end select;
    end loop;
  end OPERATOR;
```

Figure 2: Body of the OPERATOR task

```
task body PUMP is
  begin
    loop
      accept ACTIVATE;
      accept START_PUMPING;
      accept FINISH_PUMPING do
        ...      -- compute charge for this transaction
        OPERATOR.CHARGE;    -- report charge to operator
      end FINISH_PUMPING;
    end loop;
  end PUMP;

use COMMON;
task body CUSTOMER_1 is
  begin
    loop
      OPERATOR.PREPAY(c1);
      PUMP.START_PUMPING;
      PUMP.FINISH_PUMPING;
      accept CHANGE;
    end loop;
  end CUSTOMER_1;

use COMMON;
task body CUSTOMER_2 is
  begin
    loop
      OPERATOR.PREPAY(c2);
      PUMP.START_PUMPING;
      PUMP.FINISH_PUMPING;
      accept CHANGE;
    end loop;
  end CUSTOMER_2;
```

Figure 3: Bodies of the PUMP and CUSTOMER tasks

ures 5, 6, and 7, we show task expressions corresponding to the tasks in our system. These task expressions have been derived using the translation rules of [3][1], put into reduced form [5] and then simplified further. The line numbers in these figures are included for reference. The event symbols used in the task expressions are essentially those of [3], with some simplification and abbreviation. A table showing the symbols and the associated events is given in Figure 4.

To begin the analysis, we must formulate our question about the gas station in terms of the appearance of patterns of symbols in the interpreted language of the constrained expression representation of that system. Our original question was: Does a customer who prepays always get to pump gas? Prepaying is modelled by a rendezvous between the CUSTOMER and OPERATOR tasks at the entry OPERATOR.PREPAY and pumping is modelled by a rendezvous between the CUSTOMER and PUMP tasks at the entry PUMP.START_PUMPING. In the bodies of the CUSTOMER tasks, the call to OPERATOR.PREPAY is followed immediately by the call to PUMP.START_PUMPING. Therefore, the only way that a customer can prepay but fail to pump is for the CUSTOMER task to starve calling the entry PUMP.START_PUMPING.

In terms of the CEDL system, then, we may phrase our question as: Is there a behavior of the system in which one of the customer tasks starves waiting to call the PUMP.START_PUMPING entry of the pump task? The corresponding question for the constrained expression representation is: Is there a constrained

[1]The translation rules used here differ from those of [3] in one respect, the handling of *kill_rend* and *dead_rend* symbols. The translation rules of [3] use an expression *kill_exp*, which is defined to be λ if the statement S being translated does not lie in the scope of any accept statements. If S does lie in the scope of accept statements,

$$kill\_exp \equiv \bigvee_{\overline{T}' \in callers(\overline{E}')} \overline{kill\_rend}(\overline{T}', \overline{E}'),$$

where $\overline{E}'$ denotes the entries $E'_1, \ldots, E'_n$ associated with the accept statements in which S is nested and $callers(\overline{E}')$ is the set of all n-tuples $(T'_1, \ldots, T'_n)$ such that the tasks $T'_k, 1 \leq k \leq n$, are distinct and T'_k calls E'_k. In this case, the expression *kill_exp* has an alternative for each possible sequence of tasks calling the entries in which S is nested.

In the case where S lies in the scope of one or more accept statements, we have defined *kill_exp* somewhat differently. We take

$$kill\_exp \equiv \overline{kill\_rend}(\overline{E}'),$$

where, as before, $\overline{E}'$ denotes the entries $E'_1, \ldots, E'_n$ associated with the accept statements in which S is nested. We then replace the contraint of type (12) of [3] by

$$\lambda \vee \left(kill\_rend(E) \otimes \bigvee_{T \in callers(E)} dead\_rend(T, E)\right).$$

One such constraint is required for each entry E.

This modification of the translation rules produces somewhat simpler task expressions, especially when accept statements are deeply nested, and facilitates the generation of inequalities for analysis.

Symbol	*Associated event*
def(V,v)	Variable V is assigned the value v
use(V,v)	Variable V is presumed to have the value v
beg_loop(L)	Begin execution of loop L
call(T,E)	Task T calls entry E
beg_rend(T,E)	Begin rendezvous with task T at entry E
end_rend(T,E)	End rendezvous with task T at entry E
resume(T,E)	Resume task T after rendezvous at entry E
$starve_c$(T,E)	Task T starves on call to entry E
$starve_a$(E)	Task starves waiting to accept a call at entry E
kill_rend(()E)	Rendezvous at entry E is aborted
dead_rend(T,E)	Rendezvous with task T at entry E is assumed to abort
stop(T)	Execution of task T stops

In the symbols used in the task expressions, the task name CUSTOMER.i is abbreviated to Ci, PUMP is abbreviated to P, and OPERATOR is abbreviated to O. Variable and entry names are also abbreviated.

Figure 4: Event Symbols and Assoicated Events

prefix containing a $starve_c$(Ci, P.start) symbol, for i = 1 or 2?

To answer this question, we assume that there is such a prefix and generate a system of inequalities involving the number of occurrences of other event symbols in segments of that prefix. If the system is inconsistent, we conclude that no such prefix exists. If the system of inequalities is consistent, we use the inequalities in attempting to construct such a prefix and the corresponding behavior of the CEDL system.

In this case, we assume that there is a constrained prefix containing a $starve_c$(Ci, P.start) symbol, for i = 1 or 2. Since the two customer tasks are treated symmetrically in the gas station system, we may, without loss of generality, assume that a $starve_c$(C1, P.start) symbol occurs in a constrained prefix. When projected on the alphabet of the task expression τ(C1), the image of such a prefix lies in the language the expression (C1-1)*C1-4 from Figure 5.

Consider the image of such a prefix when projected on the alphabet of the task expression τ(P). We know from a constraint of the form (11) of [3] that this image must lie in the language of the expression (P-1)*(P-2 ∨ P-4 ∨ P-5). We will consider these three alternatives separately.

2.3 Analysis of the Case in Which the PUMP Task Starves at the Entry PUMP.FINISH_PUMPING

We assume that we have a constrained prefix, *s*, containing a $starve_c$(C1, P.start) symbol and a $starve_a$(P.finish) symbol. When *s* is projected on the alphabet of τ(C1), its image must lie in the language of the expression (C1-1)*C1-4, and

Ci-1 $\textit{beg\_loop}(\mathrm{Ci})\Big(\textit{call}(\mathrm{Ci}, \mathrm{O.prepay})\textit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\textit{call}(\mathrm{Ci}, \mathrm{P.start})$

$\textit{resume}(\mathrm{Ci}, \mathrm{P.start})\textit{call}(\mathrm{Ci}, \mathrm{P.finish})\textit{resume}(\mathrm{Ci}, \mathrm{P.finish})$

$\textit{beg\_rend}(\mathrm{O}, \mathrm{Ci.change})\textit{end\_rend}(\mathrm{O}, \mathrm{Ci.change})\Big)^{*}$

Ci-2 $\Big(\textit{starve}_c(\mathrm{Ci}, \mathrm{O.prepay})\textit{stop}(\mathrm{Ci})$

Ci-3 $\vee\,\textit{call}(\mathrm{Ci}, \mathrm{O.prepay})\textit{dead\_rend}(\mathrm{Ci}, \mathrm{O.prepay})\textit{stop}(\mathrm{Ci})$

Ci-4 $\vee\,\textit{call}(\mathrm{Ci}, \mathrm{O.prepay})\textit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\textit{starve}_c(\mathrm{Ci}, \mathrm{P.start})$

$\textit{stop}(\mathrm{Ci})$

Ci-5 $\vee\,\textit{call}(\mathrm{Ci}, \mathrm{O.prepay})\textit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\textit{call}(\mathrm{Ci}, \mathrm{P.start})$

$\textit{resume}(\mathrm{Ci}, \mathrm{P.start})\textit{starve}_c(\mathrm{Ci}, \mathrm{P.finish})\textit{stop}(\mathrm{Ci})$

Ci-6 $\vee\,\textit{call}(\mathrm{Ci}, \mathrm{O.prepay})\textit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\textit{call}(\mathrm{Ci}, \mathrm{P.start})$

$\textit{resume}(\mathrm{Ci}, \mathrm{P.start})\textit{call}(\mathrm{Ci}, \mathrm{P.finish})\textit{dead\_rend}(\mathrm{Ci}, \mathrm{P.finish})$

$\textit{stop}(\mathrm{Ci})$

Ci-7 $\vee\,\textit{call}(\mathrm{Ci}, \mathrm{O.prepay})\textit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\textit{call}(\mathrm{Ci}, \mathrm{P.start})$

$\textit{resume}(\mathrm{Ci}, \mathrm{P.start})\textit{call}(\mathrm{Ci}, \mathrm{P.finish})\textit{resume}(\mathrm{Ci}, \mathrm{P.finish})$

$\textit{starve}_a(\mathrm{Ci.change})\textit{stop}(\mathrm{Ci})\Big)$

Figure 5: Task Expression $\tau(\mathrm{Ci})$ Associated with the Task `CUSTOMER_i`

P-1 $$\mathit{beg\_loop}(\mathrm{P})\Big(\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})$$
$$(\bigvee_i \mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.start})\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.start}))$$
$$(\bigvee_i \mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.finish})\mathit{call}(\mathrm{P}, \mathrm{O.charge})\mathit{resume}(\mathrm{P}, \mathrm{O.charge})$$
$$\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.finish}))\Big)^*$$

P-2 $$\Big(\mathit{starve}_a(\mathrm{P.act})\mathit{stop}(\mathrm{P})$$

P-3 $$\vee\, \mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})\mathit{starve}_a(\mathrm{P.start})\mathit{stop}(\mathrm{P})$$

P-4 $$\vee\, \mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})(\bigvee_i \mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.start})$$
$$\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.start}))\mathit{starve}_a(\mathrm{P.finish})\mathit{stop}(\mathrm{P})$$

P-5 $$\vee\, \mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})(\bigvee_i \mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.start})$$
$$\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.start}))(\bigvee_i \mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.finish}))$$
$$\mathit{starve}_c(\mathrm{P}, \mathrm{O.charge})\mathit{kill\_rend}(\mathrm{P.finish})\mathit{stop}(\mathrm{P})\Big)$$

Figure 6: Task Expression $\tau(\mathrm{P})$ Associated with the Task PUMP

O-1 $\mathit{def}(\text{cus}, \text{zero})\mathit{def}(\text{current}, \perp)\mathit{def}(\text{wait}, \perp)\mathit{beg\_loop}(\text{O})\Bigg($

O-2 $\Bigg(\bigvee_i(\mathit{beg\_rend}(\text{Ci}, \text{O.prepay})\mathit{def}(\text{cus\_id}, \text{ci})(\bigvee_x \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{cus}, \text{succ(x)}))$

O-3 $\Bigg((\mathit{use}(\text{cus}, \text{one})\mathit{def}(\text{current}, \text{ci})\mathit{call}(\text{O}, \text{P.act})\mathit{resume}(\text{O}, \text{P.act}))$

O-4 $\vee(\bigvee_{\text{x}\neq\text{one}} \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{wait}, \text{ci}))\Big)\mathit{end\_rend}(\text{Ci}, \text{O.prepay})\Big)\Big)$

O-5 $\vee\Bigg(\mathit{beg\_rend}(\text{P}, \text{O.charge})\mathit{end\_rend}(\text{P}, \text{O.charge})$

O-6 $(\mathit{use}(\text{current}, \text{c1})\mathit{call}(\text{O}, \text{C1.change})\mathit{resume}(\text{O}, \text{C1.change})$

O-7 $\vee \mathit{use}(\text{current}, \text{c2})\mathit{call}(\text{O}, \text{C2.change})\mathit{resume}(\text{O}, \text{C2.change}))$

O-8 $(\bigvee_x \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{cus}, \text{pred(x)}))$

O-9 $((\bigvee_{\text{x}>\text{zero}} \mathit{use}(\text{cus}, \text{x}))(\bigvee_i \mathit{use}(\text{wait}, \text{ci})\mathit{def}(\text{current}, \text{ci}))$

$\mathit{call}(\text{O}, \text{P.act})\mathit{resume}(\text{O}, \text{P.act})$

O-10 $\vee \mathit{use}(\text{cus}, \text{zero}))\Big)\Big)^{*}$

O-11 $\Bigg(\mathit{starve}_a(\text{O.prepay})\mathit{starve}_a(\text{O.charge})\mathit{stop}(\text{O})$

O-12 $\vee(\bigvee_i \mathit{beg\_rend}(\text{Ci}, \text{O.prepay})\mathit{def}(\text{cus\_id}, \text{ci})(\bigvee_x \mathit{use}(\text{cus}, \text{x})$

$\mathit{def}(\text{cus}, \text{succ(x)}))\mathit{use}(\text{cus}, \text{one})\mathit{def}(\text{current}, \text{ci})\mathit{starve}_c(\text{O}, \text{P.act})$

$\mathit{kill\_rend}(\text{O.prepay})\mathit{stop}(\text{O}))$

Figure 7: Task Expression $\tau(\text{O})$ Associated with the Task OPERATOR

O-13 $\vee\Big(\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})\mathit{end\_rend}(\mathrm{P}, \mathrm{O.charge})\mathit{use}(\mathrm{current}, \mathrm{c1})$

$\mathit{starve}_c(\mathrm{O}, \mathrm{C1.change})\mathit{stop}(\mathrm{O})$

O-14 $\vee\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})\mathit{end\_rend}(\mathrm{P}, \mathrm{O.charge})\mathit{use}(\mathrm{current}, \mathrm{c2})$

$\mathit{starve}_c(\mathrm{O}, \mathrm{C2.change})\mathit{stop}(\mathrm{O})\Big)$

O-15 $\vee\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})\mathit{end\_rend}(\mathrm{P}, \mathrm{O.charge})$

O-16 $\Big(\mathit{use}(\mathrm{current}, \mathrm{c1})\mathit{call}(\mathrm{O}, \mathrm{C1.change})\mathit{resume}(\mathrm{O}, \mathrm{C1.change})$

O-17 $\vee\mathit{use}(\mathrm{current}, \mathrm{c2})\mathit{call}(\mathrm{O}, \mathrm{C2.change})\mathit{resume}(\mathrm{O}, \mathrm{C2.change})\Big)$

O-18 $\Big(\bigvee_{\mathrm{x}>\mathrm{zero}} \mathit{use}(\mathrm{cus}, \mathrm{x})\Big)$

O-19 $\Big(\bigvee_{i} \mathit{use}(\mathrm{wait}, \mathrm{ci})\mathit{def}(\mathrm{current}, \mathrm{ci})\Big)$

O-20 $\mathit{starve}_c(\mathrm{O}, \mathrm{P.act})\mathit{stop}(\mathrm{O})\Big)$

Figure 7: (Continued)

when s is projected on the alphabet of $\tau(\mathrm{P})$, its image must lie in the language of the expression $(\text{P-1})^*\text{P-4}$.

In each case, the symbols in a starred subexpressions must occur the same number of times. Using this observation, we obtain the following equalities relating the number of occurrences in s of symbols in these alphabets as follows. (We use the notation $|\mathit{symb}|$ to denote the number of occurrences of *symb* in s, and we ignore the *stop* and *beg_loop* symbols in this analysis.)

From the assumption that the projection of s on the alphabet of $\tau(\mathrm{C1})$ lies in the language of $(\text{C1-1})^*\text{C1-4}$, we have

$$\begin{aligned} |\mathit{starve}_c(\mathrm{C1}, \mathrm{P.start})| &= 1 & (1)\\ |\mathit{resume}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| & (2)\\ |\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| + 1 & (3) \end{aligned}$$

$$|\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| = |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| \quad (4)$$
$$|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| = |\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| \quad (5)$$
$$|\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| = |\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| \quad (6)$$
$$|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| = |\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| \quad (7)$$
$$|\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| = |\mathit{call}(\mathrm{C1}, \mathrm{P.start})|. \quad (8)$$

From the assumption that the projection of s on the alphabet of $\tau(\mathrm{P})$ lies in the language of $(\mathrm{P\text{-}1})^*\mathrm{P\text{-}4}$, we have

$$|\mathit{starve}_a(\mathrm{P.finish})| = 1 \quad (9)$$
$$|\mathit{end\_rend}(\mathrm{C1}, \mathrm{P.start})| = |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.start})| \quad (10)$$
$$|\mathit{end\_rend}(\mathrm{C2}, \mathrm{P.start})| = |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.start})| \quad (11)$$
$$\sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.start})| = |\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})| \quad (12)$$
$$|\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})| = |\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})| \quad (13)$$
$$|\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})| = \sum_i |\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.finish})| + 1 \quad (14)$$
$$\sum_i |\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.finish})| = |\mathit{resume}(\mathrm{P}, \mathrm{O.charge})| \quad (15)$$
$$|\mathit{resume}(\mathrm{P}, \mathrm{O.charge})| = |\mathit{call}(\mathrm{P}, \mathrm{O.charge})| \quad (16)$$
$$|\mathit{call}(\mathrm{P}, \mathrm{O.charge})| = \sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.finish})| \quad (17)$$
$$|\mathit{end\_rend}(\mathrm{C1}, \mathrm{P.finish})| = |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.finish})| \quad (18)$$
$$|\mathit{end\_rend}(\mathrm{C2}, \mathrm{P.finish})| = |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.finish})|. \quad (19)$$

The constraints of the form (5) of [3] give the equations

$$|\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| = |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{O.prepay})| \quad (20)$$
$$|\mathit{call}(\mathrm{C1}, \mathrm{P.start})| = |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.start})| \quad (21)$$
$$|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| = |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.finish})| \quad (22)$$
$$|\mathit{call}(\mathrm{O}, \mathrm{C1.change})| = |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| \quad (23)$$
$$|\mathit{call}(\mathrm{O}, \mathrm{P.act})| = |\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})| \quad (24)$$
$$|\mathit{call}(\mathrm{C2}, \mathrm{P.start})| = |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.start})| \quad (25)$$
$$|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| = |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.finish})| \quad (26)$$
$$|\mathit{call}(\mathrm{P}, \mathrm{O.charge})| = |\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})|. \quad (27)$$

Now consider the projection of s on the alphabet of $\tau(\mathrm{C2})$. We have the following equations and inequalities

$$|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| = |\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| - |\mathrm{C2\text{-}7}| \quad (28)$$

$$
\begin{aligned}
|\mathit{resume}(\mathrm{C2, P.finish})| &= |\mathit{call}(\mathrm{C2, P.finish})| - |\mathrm{C2\text{-}6}| && (29)\\
|\mathit{call}(\mathrm{C2, P.finish})| &= |\mathit{resume}(\mathrm{C2, P.start})| - |\mathrm{C2\text{-}5}| && (30)\\
|\mathit{resume}(\mathrm{C2, P.start})| &= |\mathit{call}(\mathrm{C2, P.start})| && (31)\\
|\mathit{call}(\mathrm{C2, P.start})| &= |\mathit{resume}(\mathrm{C2, O.prepay})| - |\mathrm{C2\text{-}4}| && (32)\\
|\mathit{resume}(\mathrm{C2, O.prepay})| &= |\mathit{call}(\mathrm{C2, O.prepay})| - |\mathrm{C2\text{-}3}| && (33)\\
|\mathit{call}(\mathrm{C2, O.prepay})| &= |\mathit{end\_rend}(\mathrm{O, C2.change})| + 1 - |\mathrm{C2\text{-}2}| && (34)\\
|\mathit{end\_rend}(\mathrm{O, C2.change})| &= |\mathit{beg\_rend}(\mathrm{O, C2.change})| && (35)\\
|\mathit{starve}_c(\mathrm{C2, O.prepay})| &= |\mathrm{C2\text{-}2}| && (36)\\
|\mathit{dead\_rend}(\mathrm{C2, O.prepay})| &= |\mathrm{C2\text{-}3}| && (37)\\
|\mathit{starve}_c(\mathrm{C2, P.start})| &= |\mathrm{C2\text{-}4}| && (38)\\
|\mathit{starve}_c(\mathrm{C2, P.finish})| &= |\mathrm{C2\text{-}5}| && (39)\\
|\mathit{dead\_rend}(\mathrm{C2, P.finish})| &= |\mathrm{C2\text{-}6}| && (40)\\
|\mathit{starve}_a(\mathrm{C2.change})| &= |\mathrm{C2\text{-}7}| && (41)\\
|\mathit{call}(\mathrm{C2, O.prepay})| &\geq |\mathrm{C2\text{-}3}| && (42)\\
|\mathit{resume}(\mathrm{C2, O.prepay})| &\geq |\mathrm{C2\text{-}4}| && (43)\\
|\mathit{resume}(\mathrm{C2, P.start})| &\geq |\mathrm{C2\text{-}5}| && (44)\\
|\mathit{call}(\mathrm{C2, P.finish})| &\geq |\mathrm{C2\text{-}6}| && (45)\\
|\mathit{resume}(\mathrm{C2, P.finish})| &\geq |\mathrm{C2\text{-}7}|, && (46)
\end{aligned}
$$

where we use the notation |C2-j| to denote the number of occurrences of the alternative C2-j in the projection of s. Since exactly one of the alternatives C2-2 through C2-7 occurs, we must also have

$$|\mathrm{C2\text{-}2}| + |\mathrm{C2\text{-}3}| + |\mathrm{C2\text{-}4}| + |\mathrm{C2\text{-}5}| + |\mathrm{C2\text{-}6}| + |\mathrm{C2\text{-}7}| = 1. \qquad (47)$$

(All variables are assumed to be nonnegative).

Finally, a constraint of the form (11) of [3] tells us that

$$
\begin{aligned}
|\mathit{starve}_a(\mathrm{P.finish})| + |\mathit{starve}_c(\mathrm{C1, P.finish})| &\leq 1 && (48)\\
|\mathit{starve}_a(\mathrm{P.finish})| + |\mathit{starve}_c(\mathrm{C2, P.finish})| &\leq 1. && (49)
\end{aligned}
$$

We use a standard branch-and-bound integer linear programming package [8] to solve this system of equations and inequalities. (For convenience, we usually choose the objective function to minimize the sum of the variables.) In this case, the package reports that the system is inconsistent. We therefore conclude that there is no constrained prefix s containing both a starve_c(C1, P.start) symbol and a starve_a(P.finish) symbol.

2.4 Analysis of the Case in Which the Pump Task Starves Calling the Entry OPERATOR.CHARGE

Now assume that our constrained prefix, s, contains a $starve_c(\mathrm{C1}, \mathrm{P.start})$ symbol and that the projection of s on the alphabet of $\tau(\mathrm{P})$ lies in the language of the expression $(\mathrm{P\text{-}1})^*\mathrm{P\text{-}5}$. This implies that s contains a $starve_c(\mathrm{P}, \mathrm{O.charge})$ symbol, and thus represents a behavior in which the pump starves waiting to call the entry OPERATOR.CHARGE.

Projection on the alphabet of $\tau(\mathrm{C1})$, yields the following inequalities, exactly as in the previous section.

$$\begin{aligned}
|starve_c(\mathrm{C1}, \mathrm{P.start})| &= 1 && (1)\\
|resume(\mathrm{C1}, \mathrm{O.prepay})| &= |call(\mathrm{C1}, \mathrm{O.prepay})| && (2)\\
|call(\mathrm{C1}, \mathrm{O.prepay})| &= |end\_rend(\mathrm{O}, \mathrm{C1.change})| + 1 && (3)\\
|end\_rend(\mathrm{O}, \mathrm{C1.change})| &= |beg\_rend(\mathrm{O}, \mathrm{C1.change})| && (4)\\
|beg\_rend(\mathrm{O}, \mathrm{C1.change})| &= |resume(\mathrm{C1}, \mathrm{P.finish})| && (5)\\
|resume(\mathrm{C1}, \mathrm{P.finish})| &= |call(\mathrm{C1}, \mathrm{P.finish})| && (6)\\
|call(\mathrm{C1}, \mathrm{P.finish})| &= |resume(\mathrm{C1}, \mathrm{P.start})| && (7)\\
|resume(\mathrm{C1}, \mathrm{P.start})| &= |call(\mathrm{C1}, \mathrm{P.start})|. && (8)
\end{aligned}$$

Our hypothesis about the projection of s on the alphabet of $\tau(\mathrm{P})$ implies that

$$\begin{aligned}
|kill\_rend(\mathrm{P.finish})| &= 1 && (9)\\
|starve_c(\mathrm{P}, \mathrm{O.charge})| &= 1 && (10)\\
\sum_i |beg\_rend(\mathrm{Ci}, \mathrm{P.finish})| &= \sum_i |end\_rend(\mathrm{Ci}, \mathrm{P.start})| && (11)\\
\sum_i |end\_rend(\mathrm{Ci}, \mathrm{P.start})| &= \sum_i |beg\_rend(\mathrm{Ci}, \mathrm{P.start})| && (12)\\
\sum_i |beg\_rend(\mathrm{Ci}, \mathrm{P.start})| &= |end\_rend(\mathrm{O}, \mathrm{P.act})| && (13)\\
|end\_rend(\mathrm{O}, \mathrm{P.act})| &= |beg\_rend(\mathrm{O}, \mathrm{P.act})| && (14)\\
|beg\_rend(\mathrm{O}, \mathrm{P.act})| &= \sum_i |end\_rend(\mathrm{Ci}, \mathrm{P.finish})| + 1 && (15)\\
\sum_i |end\_rend(\mathrm{Ci}, \mathrm{P.finish})| &= |resume(\mathrm{P}, \mathrm{O.charge})| && (16)\\
|resume(\mathrm{P}, \mathrm{O.charge})| &= |call(\mathrm{P}, \mathrm{O.charge})| && (17)\\
\sum_i |beg\_rend(\mathrm{Ci}, \mathrm{P.finish})| &\geq |kill\_rend(\mathrm{P.finish})|. && (18)
\end{aligned}$$

From the constraints of the form (5) of [3], we obtain

$$|call(\mathrm{C1}, \mathrm{O.prepay})| = |beg\_rend(\mathrm{C1}, \mathrm{O.prepay})| \qquad (19)$$

$$\begin{aligned}
|\mathit{call}(\mathrm{C1}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.start})| && (20)\\
|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.finish})| && (21)\\
|\mathit{call}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| && (22)\\
|\mathit{call}(\mathrm{O}, \mathrm{P.act})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})| && (23)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.start})| && (24)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.finish})| && (25)\\
|\mathit{call}(\mathrm{P}, \mathrm{O.charge})| &= |\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})|. && (26)
\end{aligned}$$

Exactly as in section 2.3, we obtain the following equations and inequalities by considering the projection of s on the alphabet of $\tau(\mathrm{C2})$.

$$\begin{aligned}
|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| - |\mathrm{C2\text{-}7}| && (27)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| - |\mathrm{C2\text{-}6}| && (28)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| - |\mathrm{C2\text{-}5}| && (29)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{call}(\mathrm{C2}, \mathrm{P.start})| && (30)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| - |\mathrm{C2\text{-}4}| && (31)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| - |\mathrm{C2\text{-}3}| && (32)\\
|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{C2.change})| + 1 - |\mathrm{C2\text{-}2}| && (33)\\
|\mathit{end\_rend}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| && (34)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathrm{C2\text{-}2}| && (35)\\
|\mathit{dead\_rend}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathrm{C2\text{-}3}| && (36)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{P.start})| &= |\mathrm{C2\text{-}4}| && (37)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{P.finish})| &= |\mathrm{C2\text{-}5}| && (38)\\
|\mathit{dead\_rend}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathrm{C2\text{-}6}| && (39)\\
|\mathit{starve}_a(\mathrm{C2.change})| &= |\mathrm{C2\text{-}7}| && (40)\\
|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| &\geq |\mathrm{C2\text{-}3}| && (41)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| &\geq |\mathrm{C2\text{-}4}| && (42)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| &\geq |\mathrm{C2\text{-}5}| && (43)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| &\geq |\mathrm{C2\text{-}6}| && (44)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| &\geq |\mathrm{C2\text{-}7}|, && (45)
\end{aligned}$$

and

$$|\mathrm{C2\text{-}2}| + |\mathrm{C2\text{-}3}| + |\mathrm{C2\text{-}4}| + |\mathrm{C2\text{-}5}| + |\mathrm{C2\text{-}6}| + |\mathrm{C2\text{-}7}| = 1. \qquad (46)$$

From constraints of the form (11) of [3], we have

$$\begin{aligned}
|\mathit{starve}_a(\mathrm{P.finish})| + |\mathit{starve}_c(\mathrm{C1}, \mathrm{P.finish})| &\leq 1 && (47)\\
|\mathit{starve}_a(\mathrm{P.finish})| + |\mathit{starve}_c(\mathrm{C2}, \mathrm{P.finish})| &\leq 1 && (48)\\
|\mathit{starve}_c(\mathrm{P}, \mathrm{O.charge})| + |\mathit{starve}_a(\mathrm{O.charge})| &\leq 1. && (49)
\end{aligned}$$

This system is consistent so we must examine the projection of s on the alphabet of $\tau(\mathrm{O})$.

Since we are assuming that $|starve_c(\mathrm{P}, \mathrm{O.charge})| = 1$, the last inequality implies that $|starve_a(\mathrm{O.charge})| = 0$. Hence, the projection of s on the alphabet of $\tau(\mathrm{O})$ lies in the language of the expression (O-1 $\cdots$ O-10)*(O-12 $\vee$ (O-13 $\vee$ O-14) $\vee$ (O-15(O-16 $\vee$ O-17)O-18 $\cdots$ O-20)). We will treat each of these cases in turn.

2.4.1 The (O-1 $\cdots$ O-10)*O-12 Case

Assume first that the projection of s on the alphabet of $\tau(\mathrm{O})$ lies in the language of the expression (O-1 $\cdots$ O-10)*O-12. Then s contains a $starve_c(\mathrm{O}, \mathrm{P.act})$ symbol, and so represents a behavior in which the task OPERATOR starves calling the entry PUMP.ACTIVATE. We have

$$|kill\_rend(\mathrm{O.prepay})| = 1 \tag{50}$$
$$|starve_c(\mathrm{O}, \mathrm{P.act})| = 1. \tag{51}$$

From the constraints involving *kill_rend* and *dead_rend* symbols, we have

$$|kill\_rend(\mathrm{O.prepay})| - \sum_i |dead\_rend(\mathrm{Ci}, \mathrm{O.prepay}) = 0, \tag{52}$$

and we see from our hypothesis on s that

$$|dead\_rend(\mathrm{C1}, \mathrm{O.prepay})| = 0. \tag{53}$$

Similarly, we have

$$|krendP.finish| - \sum_i |dead\_rend(\mathrm{Ci}, \mathrm{P.finish})| = 0 \tag{54}$$
$$|dead\_rend(\mathrm{C1}, \mathrm{P.finish})| = 0. \tag{55}$$

The integer linear programming package reports that this system is inconsistent, and we conclude that no such constrained prefix exists.

2.4.2 The (O-1 $\cdots$ O-10)*(O-13 $\vee$ O-14) Case

Assume now that the projection of s on the alphabet of $\tau(\mathrm{O})$ lies in the language of the expression (O-1 $\cdots$)-10)*(O-13 $\vee$ O-14). Then our constrained prefix contains a $starve_c(\mathrm{O}, \mathrm{Ci.change})$ symbol, and so represents a behavior in which the task OPERATOR starves calling one of the entries CUSTOMER_i.CHANGE.

We have

$$|starve_c(\mathrm{O}, \mathrm{C1.change}) = |\text{O-13}| \tag{50}$$
$$|starve_c(\mathrm{O}, \mathrm{C2.change}) = |\text{O-14}| \tag{51}$$
$$|end\_rend(\mathrm{P}, \mathrm{O.charge})| = \sum_i |call(\mathrm{O}, \mathrm{Ci.change})| + 1 \tag{52}$$
$$|\text{O-13}| + |\text{O-14}| = 1 \tag{53}$$

from $\tau(\mathrm{O})$, where $|\text{O-j}|$ denotes the number of occurrences of the alternative O-j in the projection of s.

From the constraints of the form (11) of [3], we have

$$\begin{aligned}
|\mathit{starve}_a(\text{C1.change})| + |\mathit{starve}_c(\text{O}, \text{C1.change})| &\leq 1 &(54)\\
|\mathit{starve}_a(\text{C2.change})| + |\mathit{starve}_c(\text{O}, \text{C2.change})| &\leq 1. &(55)
\end{aligned}$$

The integer linear programming package reports that this system of equations and inequalities is inconsistent, and we conclude that no such constrained prefix s exists.

2.4.3 The (O-1 $\cdots$ O-10)*O-15(O-16 $\vee$ O-17)O-18 $\cdots$ O-20 Case

In this case, s contains a $\mathit{starve}_c(\text{O}, \text{P.act})$ symbol, and represents a behavior in which the task OPERATOR starves calling the entry PUMP.ACTIVATE following a rendezvous at the entry OPERATOR.CHARGE. From $\tau(\text{O})$, we have

$$\begin{aligned}
|\mathit{starve}_c(\text{O}, \text{P.act})| &= 1 &(50)\\
|\mathit{resume}(\text{O}, \text{C1.change})| &= |\mathit{call}(\text{O}, \text{C1.change})| &(51)\\
|\mathit{resume}(\text{O}, \text{C2.change})| &= |\mathit{call}(\text{O}, \text{C2.change})| &(52)\\
\sum_i |\mathit{call}(\text{O}, \text{Ci.change})| &= |\mathit{end\_rend}(\text{P}, \text{O.charge})| &(53)\\
|\mathit{end\_rend}(\text{P}, \text{O.charge})| &= |\mathit{beg\_rend}(\text{P}, \text{O.charge})| &(54)\\
\sum_i |\mathit{resume}(\text{O}, \text{Ci.change})| &\geq |\mathit{starve}_c(\text{O}, \text{P.act})|. &(55)
\end{aligned}$$

From a constraint of the form (5) of [3], we have

$$|\mathit{call}(\text{O}, \text{C2.change})| = |\mathit{beg\_rend}(\text{O}, \text{C2.change})|. \qquad (56)$$

The integer linear programming packages finds a solution to this system. This solution corresponds to a string representing a behavior in which the task CUSTOMER_1 prepays and starves calling the entry PUMP.START_PUMPING and CUSTOMER_2 pumps gas once, receives change, prepays and starts pumping again and then calls PUMP.FINISH_PUMPING. This last rendezvous cannot be completed, however, because the PUMP task starves trying to call OPERATOR.CHARGE. An examination of the task expressions, however, shows that such a string cannot be a constrained prefix. From the task expression $\tau(\text{P})$ we see that, in any prefix of astring in the language of the system expression,

$$|\mathit{resume}(\text{P}, \text{O.charge})| \leq |\mathit{end\_rend}(\text{O}, \text{P.act})| \leq |\mathit{resume}(\text{P}, \text{O.charge})| + 1,$$

with equality holding on the left if the last symbol of either of these two types is a $\mathit{resume}(\text{P}, \text{O.charge})$ and equality holding on the right if the last symbol is an *end_rend*(O, P.act). (This is just a statement of the fact that rendezvous at PUMP.ACTIVATE and OPERATOR.CHARGE alternate in any behavior of the system.) Our hypothesis concerning the projection of s on the alphabet of $\tau(\mathrm{O})$ and constraints of type (6) of [3] imply that the last symbol in s of either of these types is a *resume*(P, O.charge), so we have

$$|\mathit{resume}(\mathrm{P}, \mathrm{O.charge})| = |\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})|. \tag{57}$$

The integer linear programming package reports that this system is inconsistent, and we conclude that no such contrained prefix s exists. This completes the analysis of the case in which the PUMP task starves calling the entry OPERATOR.CHARGE.

2.5 Analysis of the Case in Which the Pump Task Starves at the Entry PUMP.ACTIVATE

Let s be a constrained prefix containing a starve_c(C1, P.start) symbol, and consider the projection of s on the alphabet of $\tau(\mathrm{P})$. We have shown that this projection does not lie in the language of the expression (P-1)*(P-3 $\vee$ P-4 $\vee$ P-5), so it must lie in the language of (P-1)*P-2. This implies that s contains a starve_a(P.act) symbol, and thus represents a behavior in which the pump starves at the entry PUMP.ACTIVATE.

Projecting s on the alphabet of $\tau(\mathrm{C1})$, we obtain as usual

$$\begin{aligned}
|\mathit{starve}_c(\mathrm{C1}, \mathrm{P.start})| &= 1 && (1)\\
|\mathit{resume}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| && (2)\\
|\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| + 1 && (3)\\
|\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| && (4)\\
|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| && (5)\\
|\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| && (6)\\
|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| && (7)\\
|\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| &= |\mathit{call}(\mathrm{C1}, \mathrm{P.start})|. && (8)
\end{aligned}$$

Projecting on the alphabet of $\tau(\mathrm{P})$, we have

$$\begin{aligned}
|\mathit{starve}_a(\mathrm{P.act})| &= 1 && (9)\\
\sum_i |\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.finish})| &= |\mathit{resume}(\mathrm{P}, \mathrm{O.charge})| && (10)\\
|\mathit{resume}(\mathrm{P}, \mathrm{O.charge})| &= |\mathit{call}(\mathrm{P}, \mathrm{O.charge})| && (11)\\
|\mathit{call}(\mathrm{P}, \mathrm{O.charge})| &= \sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.finish})| && (12)\\
\sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.finish})| & \quad \sum_i |\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.start})| && (13)
\end{aligned}$$

$$\sum_i |\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.start})| = \sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.start})| \quad (14)$$
$$\sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.start})| = |\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})| \quad (15)$$
$$|\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})| = |\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})|. \quad (16)$$

From $\tau(\mathrm{C2})$, we obtain, as in the preceding sections,

$$\begin{aligned}
|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| - |\mathrm{C2\text{-}7}| && (17)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| - |\mathrm{C2\text{-}6}| && (18)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| - |\mathrm{C2\text{-}5}| && (19)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{call}(\mathrm{C2}, \mathrm{P.start})| && (20)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| - |\mathrm{C2\text{-}4}| && (21)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| - |\mathrm{C2\text{-}3}| && (22)\\
|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{C2.change})| + 1 - |\mathrm{C2\text{-}2}| && (23)\\
|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{C2.change})| && (24)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathrm{C2\text{-}2}| && (25)\\
|\mathit{dead\_rend}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathrm{C2\text{-}3}| && (26)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{P.start})| &= |\mathrm{C2\text{-}4}| && (27)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{P.finish})| &= |\mathrm{C2\text{-}5}| && (28)\\
|\mathit{dead\_rend}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathrm{C2\text{-}6}| && (29)\\
|\mathit{starve}_a(\mathrm{C2.change})| &= |\mathrm{C2\text{-}7}| && (30)\\
|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| &\geq |\mathrm{C2\text{-}3}| && (31)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| &\geq |\mathrm{C2\text{-}4}| && (32)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| &> |\mathrm{C2\text{-}5}| && (33)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| &\geq |\mathrm{C2\text{-}6}| && (34)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| &\geq |\mathrm{C2\text{-}7}| && (35)
\end{aligned}$$

and

$$|\mathrm{C2\text{-}2}| + |\mathrm{C2\text{-}3}| + |\mathrm{C2\text{-}4}| + |\mathrm{C2\text{-}5}| + |\mathrm{C2\text{-}6}| + |\mathrm{C2\text{-}7}| = 1. \quad (36)$$

The constraints of the form (5) of [3] give

$$\begin{aligned}
|\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{O.prepay})| && (37)\\
|\mathit{call}(\mathrm{C1}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.start})| && (38)\\
|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.finish})| && (39)\\
|\mathit{call}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| && (40)\\
|\mathit{call}(\mathrm{O}, \mathrm{P.act})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})| && (41)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.start})| && (42)
\end{aligned}$$

$$
\begin{aligned}
|call(\text{C2}, \text{P.finish})| &= |beg\_rend(\text{C2}, \text{P.finish})| && (43)\\
|call(\text{P}, \text{O.charge})| &= |beg\_rend(\text{P}, \text{O.charge})| && (44)\\
|call(\text{C2}, \text{O.prepay})| &= |beg\_rend(\text{C2}, \text{O.prepay})| && (45)\\
|call(\text{O}, \text{C2.change})| &= |beg\_rend(\text{O}, \text{C2.change})|. && (46)
\end{aligned}
$$

The constraints of the form (11) of [3] tell us that

$$
\begin{aligned}
|starve_a(\text{P.act})| + |starve_c(\text{O}, \text{P.act})| &\leq 1 && (47)\\
|starve_c(\text{C1}, \text{O.prepay})| + |starve_a(\text{O.prepay})| &\leq 1 && (48)\\
|starve_c(\text{C2}, \text{O.prepay})| + |starve_a(\text{O.prepay})| &\leq 1 && (49)\\
|starve_a(\text{C1.change})| + |starve_c(\text{O}, \text{C1.change})| &\leq 1 && (50)\\
|starve_a(\text{C2.change})| + |starve_c(\text{O}, \text{C2.change})| &\leq 1. && (51)
\end{aligned}
$$

Since we have $|starve_a(\text{P.act})| = 1$, we conclude that $|starve_c(\text{O}, \text{P.act})| = 0$. This implies that the projection of s onto the alphabet of $\tau(\text{O})$ lies in the language of the expression $(\text{O-1} \cdots \text{O-10})^*(\text{O-11} \vee \text{O-13} \vee \text{O-14})$. We consider each of these alternatives separately.

2.5.1 The $(\text{O-1} \cdots \text{O-10})^*\text{O-11}$ Case

From $\tau(\text{O})$, we have

$$
\begin{aligned}
|starve_a(\text{O.prepay})| &= 1 && (52)\\
|starve_a(\text{O.charge})| &= 1, && (53)
\end{aligned}
$$

and we see that, in this case, s represents a behavior in which the OPERATOR task starves at the `select` statement.

We also have

$$|kill\_rend(\text{O.prepay})| = 0. \quad (54)$$

The *kill_rend-dead_rend* constraints give

$$|kill\_rend(\text{O.prepay})| - \sum_i |dead\_rend(\text{Ci}, \text{O.prepay})| = 0. \quad (55)$$

The integer linear programming package finds a solution to this system, corresponding to a behavior in which each customer task completes a rendezvous at the entry OPERATOR.PREPAY, but the pump task starves waiting for a rendezvous at the entry PUMP.ACTIVATE. Both customer tasks then starve waiting for rendezvous at PUMP.START_PUMPING.

An examination of the CEDL programs indicates that such a behavior should be impossible, since the first rendezvous at OPERATOR.PREPAY cannot be completed without a (nested) rendezvous at PUMP.ACTIVATE. The problem is that our system of inequalities does not reflect the correct branching based on the value of the variable cus. The easiest way to resolve this problem is to establish another inequality to be added to the system.

Lemma 2.1 *Let s be a constrained prefix. Then*

$$\sum_i |resume(\text{Ci}, \text{O.prepay})| \leq |resume(\text{O}, \text{P.act})| + 1.$$

Proof. From the task expresions for the customer tasks, we see that

$$\sum_i |call(\text{Ci}, \text{P.start})| \leq \sum_i |resume(\text{Ci}, \text{O.prepay})| \leq \sum_i |call(\text{Ci}, \text{P.start})| + 2$$

and, from the constraints and the task expression for the pump,

$$\sum_i |call(\text{Ci}, \text{P.start})| \leq |resume(\text{O}, \text{P.act})| \leq \sum_i |call(\text{Ci}, \text{P.start})| + 1$$

in any prefix of s. It follows that

$$\sum_i |resume(\text{Ci}, \text{O.prepay})| \leq |resume(\text{O}, \text{P.act})| + 2,$$

with equality holding only in a prefix for which $\sum_i |resume(\text{Ci}, \text{O.prepay})| = \sum_i |call(\text{Ci}, \text{P.start})| + 2$.

Assume that equality holds for s, so s represents a behavior in which each customer prepays once more than it starts pumping. Consider the last two $beg\_rend(\text{Ci}, \text{O.prepay})$ symbols in s. If a $resume(\text{O}, \text{P.act})$ symbol followed either of these, we would have $\sum_i |resume(\text{Ci}, \text{O.prepay})| \geq |resume(\text{O}, \text{P.act})| + 3$ in the prefix of s ending at the first of these $beg\_rend(\text{Ci}, \text{O.prepay})$ symbols, and we have seen that this is impossible. It follows that the $def(\text{cus}, \_)$ symbol following the first of these $beg\_rend(\text{Ci}, \text{O.prepay})$ symbols must be a $def(\text{cus}, \text{two})$.

It is an easy exercise to show that, in any constrained prefix not ending in a $beg\_rend(\text{Ci}, \text{O.prepay})$ symbol, a $def(\text{cus\_id}, \text{ci})$ symbol, or a $use(\text{cus}, \_)$ symbol, the value of cus in the last $use(\text{cus}, \_)$ symbol is $\sum_i |beg\_rend(\text{Ci}, \text{O.prepay})| - \sum_i |resume(\text{O}, \text{Ci.change})|$, where we abuse the notation by equating the value of an enumeration type and the corresponding integer. (This is just the verification that the variable cus keeps track of the number of customers who have currently prepaid but not yet received change.) So, in the prefix ending just before the first of our two $beg\_rend(\text{Ci}, \text{O.prepay})$ symbols, we must have

$$\sum_i |resume(\text{Ci}, \text{O.prepay})| - \sum_i |resume(\text{O}, \text{Ci.change})| = 1.$$

We conclude that there must be a $resume(\text{O}, \text{Ci.change})$ symbol between the $beg\_rend(\text{Ci}, \text{O.prepay})$ symbols.

Since this occurs with cus $= 2$, it must be followed by a $call(\text{O}, \text{P.act})$ symbol or a $starve_c(\text{O}, \text{P.act})$ symbol before the last $beg\_rend(\text{Ci}, \text{O.prepay})$. Since another prepay rendezvous occurs, the $starve_c(\text{O}, \text{P.act})$ is impossible.

Thus, a *resume*(O, P.act) symbol must occur somewhere between the last two *beg_rend*(Ci, O.prepay) symbols in s. This is a contradiction. ∎

So the lemma tells us that

$$\sum_i |\textit{resume}(\text{Ci}, \text{O.prepay})| \leq |\textit{resume}(\text{O}, \text{P.act})| + 1, \qquad (56)$$

and a constraint of type (6) gives

$$|\textit{resume}(\text{O}, \text{P.act})| = |\textit{end\_rend}(\text{O}, \text{P.act})|. \qquad (57)$$

The integer linear programming package reports that this system is inconsistent, and we conclude that no such constrained prefix exists.

2.5.2 The (O-1 ··· O-10)*O-13 Case

From $\tau(\text{O})$, we have

$$\begin{aligned}
|\textit{starve}_c(\text{O}, \text{C1.change})| &\quad 1 && (52)\\
|\textit{end\_rend}(\text{P}, \text{O.charge})| &> 1 && (53)\\
|\textit{end\_rend}(\text{P}, \text{O.charge})| &= |\textit{beg\_rend}(\text{P}, \text{O.charge})| && (54)\\
|\textit{beg\_rend}(\text{P}, \text{O.charge})| &= \sum_i |\textit{resume}(\text{O}, \text{Ci.change})| + 1 && (55)\\
|\textit{resume}(\text{O}, \text{C1.change})| &= |\textit{call}(\text{O}, \text{C1.change})| && (56)\\
|\textit{resume}(\text{O}, \text{C2.change})| &= |\textit{call}(\text{O}, \text{C2.change})|. && (57)
\end{aligned}$$

We also have

$$|\textit{kill\_rend}(\text{P.finish})| = 0, \qquad (58)$$

and

$$\begin{aligned}
|\textit{kill\_rend}(\text{P.finish})| - \sum_i |\textit{dead\_rend}(\text{Ci}, \text{P.finish})| &= 0 && (59)\\
|\textit{dead\_rend}(\text{C1}; \text{P.finish}, |)| &= 0. && (60)
\end{aligned}$$

When we apply the integer linear programming package to this system, having set the objective function to minimize the sum of the variables, it finds an optimal solution. Examining this solution, we see that $|\textit{starve}_a(\text{C2.change})| = 1$. If this solution actually corresponds to a constrained prefix, that prefix represents a behavior in which the task CUSTOMER_1 starves calling the entry PUMP.START_PUMPING, the task CUSTOMER_2 starves waiting for a rendezvous at its entry CUSTOMER_2.CHANGE, the task PUMP starves waiting for a rendezvous at its entry PUMP.ACTIVATE, and the task OPERATOR starves calling the entry CUSTOMER_1.CHANGE. In terms of the gas station modelled by the CEDL system, we have a situation where the operator is trying to give change to the first customer, who is waiting to pump gas, and the second customer is waiting to get change, already having pumped.

It is not hard to construct a constrained prefix representing such a behavior. One such is

$$\begin{aligned}
&\textit{beg\_loop}(\mathrm{C1})\textit{beg\_loop}(\mathrm{P})\textit{beg\_loop}(\mathrm{C2})\textit{def}(\mathrm{cus}, \mathrm{zero})\textit{def}(\mathrm{current}, \bot)\\
&\textit{def}(\mathrm{wait}, \bot)\textit{beg\_loop}(\mathrm{O})\textit{call}(\mathrm{C1}, \mathrm{O.prepay})\textit{beg\_rend}(\mathrm{C1}, \mathrm{O.prepay})\\
&\textit{def}(\mathrm{cus\_id}, \mathrm{c1})\textit{use}(\mathrm{cus}, \mathrm{zero})\textit{def}(\mathrm{cus}, \mathrm{one})\textit{use}(\mathrm{cus}, \mathrm{one})\\
&\textit{def}(\mathrm{current}, \mathrm{c1})\textit{call}(\mathrm{O}, \mathrm{P.act})\textit{beg\_rend}(\mathrm{O}, \mathrm{P.act})\textit{end\_rend}(\mathrm{O}, \mathrm{P.act})\\
&\textit{resume}(\mathrm{O}, \mathrm{P.act})\textit{end\_rend}(\mathrm{C1}, \mathrm{O.prepay})\textit{resume}(\mathrm{C1}, \mathrm{O.prepay})\\
&\textit{call}(\mathrm{C2}, \mathrm{O.prepay})\textit{beg\_rend}(\mathrm{C2}, \mathrm{O.prepay})\textit{def}(\mathrm{cus_id}, \mathrm{c2})\textit{use}(\mathrm{cus}, \mathrm{one})\\
&\textit{def}(\mathrm{cus}, \mathrm{two})\textit{use}(\mathrm{cus}, \mathrm{two})\textit{def}(\mathrm{wait}, \mathrm{c2})\textit{end\_rend}(\mathrm{C2}, \mathrm{O.prepay})\\
&\textit{resume}(\mathrm{C2}, \mathrm{O.prepay})\textit{call}(\mathrm{C2}, \mathrm{P.start})\textit{beg\_rend}(\mathrm{C2}, \mathrm{P.start})\\
&\textit{end\_rend}(\mathrm{C2}, \mathrm{P.start})\textit{resume}(\mathrm{C2}, \mathrm{P.start})\textit{call}(\mathrm{C2}, \mathrm{P.finish})\\
&\textit{beg\_rend}(\mathrm{C2}, \mathrm{P.finish})\textit{call}(\mathrm{P}, \mathrm{O.charge})\textit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})\\
&\textit{end\_rend}(\mathrm{P}, \mathrm{O.charge})\textit{resume}(\mathrm{P}, \mathrm{O.charge})\textit{end\_rend}(\mathrm{C2}, \mathrm{P.finish})\\
&\textit{resume}(\mathrm{C2}, \mathrm{P.finish})\textit{use}(\mathrm{current}, \mathrm{c1})\textit{starve}_c(\mathrm{O}, \mathrm{C1.change})\\
&\textit{starve}_c(\mathrm{C1}, \mathrm{P.start})\textit{starve}_a(\mathrm{P.act})\textit{starve}_a(\mathrm{C2.change})\textit{stop}(\mathrm{C1})\\
&\textit{stop}(\mathrm{C2})\textit{stop}(\mathrm{P})\textit{stop}(\mathrm{O}).
\end{aligned}$$

(This constrained prefix corresponds exactly to the optimal solution found by the integer linear programming package.)

2.5.3 The (O-1 ⋯ O-10)* O-14 Case

From $\tau(\mathrm{O})$, we have

$$\begin{aligned}
|\textit{starve}_c(\mathrm{O}, \mathrm{C2.change})| &= 1 && (52)\\
|\textit{end\_rend}(\mathrm{P}, \mathrm{O.charge})| &\geq 1 && (53)\\
|\textit{end\_rend}(\mathrm{P}, \mathrm{O.charge})| &= |\textit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})| && (54)\\
|\textit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})| &= \sum_i |\textit{resume}(\mathrm{O}, \mathrm{Ci.change})| + 1 && (55)\\
|\textit{resume}(\mathrm{O}, \mathrm{C1.change})| &= |\textit{call}(\mathrm{O}, \mathrm{C1.change})| && (56)\\
|\textit{resume}(\mathrm{O}, \mathrm{C2.change})| &= |\textit{call}(\mathrm{O}, \mathrm{C2.change})|. && (57)
\end{aligned}$$

We also have

$$|\textit{kill\_rend}(\mathrm{P.finish})| = 0, \qquad (58)$$

and

$$|\textit{kill\_rend}(\mathrm{P.finish})| - \sum_i |\textit{dead\_rend}(\mathrm{Ci}, \mathrm{P.finish})| = 0 \qquad (59)$$

```
task body PUMP is
  begin
    loop
      accept ACTIVATE;
      accept START_PUMPING;
      accept FINISH_PUMPING do
        ...     -- compute charge for this transaction
      end FINISH_PUMPING;
      OPERATOR.CHARGE;     -- report charge to operator
    end loop;
  end PUMP;
```

Figure 8: Revised Version of the Body of the PUMP Task

$$|dead\_rend(\mathrm{C1}; \mathrm{P.finish}, |) \quad = \quad 0. \qquad (60)$$

The integer linear programming package reports that this system is inconsistent, and we conclude that no such s exists.

3 A Revised Two-Customer System

Our analysis of the original system has uncovered a flaw in the design: it is possible for a customer to prepay but never get to pump gas. Following Helmbold and Luckham [7], we attempt to correct this flaw by changing the order of the calls to the CHANGE and ACTIVATE entries in the accept CHARGE alternative of the OPERATOR task and moving the call to the operator out of the accept FINISH_PUMPING statement in the body of the PUMP task. The revised task bodies are shown in Figures 8 and 9. The corresponding task expressions are given in Figures 10, 11, and 12. (Although the task bodies for the customer tasks are unchanged, the reduced and simplified task expressions for the customers must be modified to reflect the fact that pump task can no longer starve during a rendezvous at the entry PUMP.FINISH_PUMPING. This eliminates the alternative Ci-6 in the original task expressions.)

We now analyze the revised system to show that this flaw has indeed been corrected. As before, we assume that there is a constrained prefix containing a $starve_c$(C1, P.start) symbol and conclude that the projection of such a prefix on the alphabet of τ(P) must lie in the language of the expression (P-1)*(P-2 $\vee$ P-4 $\vee$ P-5).

```
use COMMON;
task body OPERATOR is
  CUSTOMERS : COUNTER := zero;
  CURRENT, WAITING : C_NAME;
  begin
    loop
      select
        accept PREPAY(CUSTOMER_ID : in C_NAME) do
          CUSTOMERS := COUNTER'succ(CUSTOMERS);
          if CUSTOMERS = one then       -- if no previous customer
                                        -- is waiting
            CURRENT := CUSTOMER_ID;     -- mark this one as current
            PUMP.ACTIVATE;              -- and activate the pump
           else
            WAITING := CUSTOMER_ID;      -- otherwise, mark this one
                                         -- as next in line
          end if;
        end PREPAY;
      or
        accept CHARGE;
        if CUSTOMERS > one then          -- if another customer is
                                         -- waiting,
          PUMP.ACTIVATE;                 -- activate the pump
        end if;
        if CURRENT = c1 then
          CUSTOMER_1.CHANGE;
         else
          CUSTOMER_2.CHANGE;
        end if;
        CUSTOMERS := COUNTER'pred(CUSTOMERS);
        if CUSTOMERS > zero then     -- if another customer is
          CURRENT := WAITING;        -- waiting, promote that one
                                     -- to be current
        end if;
      end select;
    end loop;
  end OPERATOR;
```

Figure 9: Revised Version of the Body of the OPERATOR task

Ci-1 $\mathit{beg\_loop}(\mathrm{Ci})\Big(\mathit{call}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{call}(\mathrm{Ci}, \mathrm{P.start})$

$\mathit{resume}(\mathrm{Ci}, \mathrm{P.start})\mathit{call}(\mathrm{Ci}, \mathrm{P.finish})\mathit{resume}(\mathrm{Ci}, \mathrm{P.finish})$

$\mathit{beg\_rend}(\mathrm{O}, \mathrm{Ci.change})\mathit{end\_rend}(\mathrm{O}, \mathrm{Ci.change})\Big)^{*}$

Ci-2 $\Big(\mathit{starve}_c(\mathrm{Ci}, \mathrm{O.prepay})\mathit{stop}(\mathrm{Ci})$

Ci-3 $\vee\,\mathit{call}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{dead\_rend}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{stop}(\mathrm{Ci})$

Ci-4 $\vee\,\mathit{call}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{starve}_c(\mathrm{Ci}, \mathrm{P.start})$

$\mathit{stop}(\mathrm{Ci})$

Ci-5 $\vee\,\mathit{call}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{call}(\mathrm{Ci}, \mathrm{P.start})$

$\mathit{resume}(\mathrm{Ci}, \mathrm{P.start})\mathit{starve}_c(\mathrm{Ci}, \mathrm{P.finish})\mathit{stop}(\mathrm{Ci})$

Ci-6 $\vee\,\mathit{call}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{resume}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{call}(\mathrm{Ci}, \mathrm{P.start})$

$\mathit{resume}(\mathrm{Ci}, \mathrm{P.start})\mathit{call}(\mathrm{Ci}, \mathrm{P.finish})\mathit{resume}(\mathrm{Ci}, \mathrm{P.finish})$

$\mathit{starve}_a(\mathrm{Ci.change})\mathit{stop}(\mathrm{Ci})\Big)$

Figure 10: Task Expression $\tau(\mathrm{Ci})$ Associated with the Revised Version of the Task `CUSTOMER_i`

P-1 $$beg\_loop(\mathrm{P})\Big(beg\_rend(\mathrm{O}, \mathrm{P.act})end\_rend(\mathrm{O}, \mathrm{P.act})$$

$$(\bigvee_i beg\_rend(\mathrm{Ci}, \mathrm{P.start})end\_rend(\mathrm{Ci}, \mathrm{P.start}))$$

$$(\bigvee_i beg\_rend(\mathrm{Ci}, \mathrm{P.finish})end\_rend(\mathrm{Ci}, \mathrm{P.finish}))$$

$$call(\mathrm{P}, \mathrm{O.charge})resume(\mathrm{P}, \mathrm{O.charge})\Big)^*$$

P-2 $$\Big(starve_a(\mathrm{P.act})stop(\mathrm{P})$$

P-3 $$\vee beg\_rend(\mathrm{O}, \mathrm{P.act})end\_rend(\mathrm{O}, \mathrm{P.act})starve_a(\mathrm{P.start})stop(\mathrm{P})$$

P-4 $$\vee beg\_rend(\mathrm{O}, \mathrm{P.act})end\_rend(\mathrm{O}, \mathrm{P.act})(\bigvee_i beg\_rend(\mathrm{Ci}, \mathrm{P.start})$$

$$end\_rend(\mathrm{Ci}, \mathrm{P.start}))starve_a(\mathrm{P.finish})stop(\mathrm{P})$$

P-5 $$\vee beg\_rend(\mathrm{O}, \mathrm{P.act})end\_rend(\mathrm{O}, \mathrm{P.act})(\bigvee_i beg\_rend(\mathrm{Ci}, \mathrm{P.start})$$

$$end\_rend(\mathrm{Ci}, \mathrm{P.start}))(\bigvee_i beg\_rend(\mathrm{Ci}, \mathrm{P.finish})$$

$$end\_rend(\mathrm{Ci}, \mathrm{P.finish}))starve_c(\mathrm{P}, \mathrm{O.charge})stop(\mathrm{P})\Big)$$

Figure 11: Task Expression $\tau(\mathrm{P})$ Associated with the Revised Version of the Task `PUMP`

O-1 $\mathit{def}(\text{cus}, \text{zero})\mathit{def}(\text{current}, \perp)\mathit{def}(\text{wait}, \perp)\mathit{beg\_loop}(\text{O})\Bigg($

O-2 $\Bigg(\bigvee_i (\mathit{beg\_rend}(\text{Ci}, \text{O.prepay})\mathit{def}(\text{cus\_id}, \text{ci})(\bigvee_x \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{cus}, \text{succ(x)}))$

O-3 $\Big((\mathit{use}(\text{cus}, \text{one})\mathit{def}(\text{current}, \text{ci})\mathit{call}(\text{O}, \text{P.act})\mathit{resume}(\text{O}, \text{P.act}))$

O-4 $\vee(\bigvee_{x \neq one} \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{wait}, \text{ci}))\Big)\mathit{end\_rend}(\text{Ci}, \text{O.prepay})\Big)\Bigg)$

O-5 $\vee\Bigg(\mathit{beg\_rend}(\text{P}, \text{O.charge})\mathit{end\_rend}(\text{P}, \text{O.charge})$

O-6 $((\bigvee_{x > one} \mathit{use}(\text{cus}, \text{x})\mathit{call}(\text{O}, \text{P.act})\mathit{resume}(\text{O}, \text{P.act})) \vee \bigvee_{x \neq one} \mathit{use}(\text{cus}, \text{x}))$

O-7 $(\mathit{use}(\text{current}, \text{c1})\mathit{call}(\text{O}, \text{C1.change})\mathit{resume}(\text{O}, \text{C1.change})$

O-8 $\vee\mathit{use}(\text{current}, \text{c2})\mathit{call}(\text{O}, \text{C2.change})\mathit{resume}(\text{O}, \text{C2.change}))$

O-9 $(\bigvee_x \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{cus}, \text{pred(x)}))$

O-10 $((\bigvee_{x > zero} \mathit{use}(\text{cus}, \text{x}))(\bigvee_i \mathit{use}(\text{wait}, \text{ci})\mathit{def}(\text{current}, \text{ci}))$

O-11 $\vee\mathit{use}(\text{cus}, \text{zero}))\Big)\Bigg)^*$

O-12 $\Bigg(\mathit{starve}_a(\text{O.prepay})\mathit{starve}_a(\text{O.charge})\mathit{stop}(\text{O})$

O-13 $\vee(\bigvee_i \mathit{beg\_rend}(\text{Ci}, \text{O.prepay})\mathit{def}(\text{cus\_id}, \text{ci})(\bigvee_x \mathit{use}(\text{cus}, \text{x})$

$\mathit{def}(\text{cus}, \text{succ(x)}))\mathit{use}(\text{cus}, \text{one})\mathit{def}(\text{current}, \text{ci})\mathit{starve}_c(\text{O}, \text{P.act})$

$\mathit{kill\_rend}(\text{O.prepay})\mathit{stop}(\text{O}))$

Figure 12: Task Expression $\tau(\text{O})$ Associated with the Revised Version of the Task OPERATOR

O-14 $\vee$ *beg_rend*(P, O.charge)*end_rend*(P, O.charge)

O-15 $\Big(\big(\bigvee_{\text{x>one}}$ *use*(cus, x)$)$*starve$_c$*(O, P.act)*stop*(O)

O-16 $\vee$ *beg_rend*(P, O.charge)*end_rend*(P, O.charge)$\big(\big(\bigvee_{\text{x>one}}$ *use*(cus, x)$)$

call(O, P.act)*resume*(O, P.act) $\vee \bigvee_{\text{x}\leq\text{one}}$ *use*(cus, x)$\big)$

O-17 $\Big($*use*(current, c1)*starve$_c$*(O, C1.change)*stop*(O)

O-18 $\vee$ *use*(current, c2)*starve$_c$*(O, C2.change)*stop*(O)$\Big)\Big)$

Figure 12: (Continued)

3.1 Analysis of the Case in Which the Pump Task Starves at the Entry PUMP.FINISH_PUMPING

Let s be a constrained prefix containing a *starve$_c$*(C1, P.finish) symbol, and assume that the projection of s on the alphabet of τ(P) lies in the language of (P-1)*P-4.

As before, we obtain the folowing equalities from the projection of s on the alphabet of τ(C1).

$$\begin{aligned}
|\mathit{starve}_c(\text{C1, P.start})| &= 1 & (1)\\
|\mathit{resume}(\text{C1, O.prepay})| &= |\mathit{call}(\text{C1, O.prepay})| & (2)\\
|\mathit{call}(\text{C1, O.prepay})| &= |\mathit{end\_rend}(\text{O, C1.change})| + 1 & (3)\\
|\mathit{end\_rend}(\text{O, C1.change})| &= |\mathit{beg\_rend}(\text{O, C1.change})| & (4)\\
|\mathit{beg\_rend}(\text{O, C1.change})| &= |\mathit{resume}(\text{C1, P.finish})| & (5)\\
|\mathit{resume}(\text{C1, P.finish})| &= |\mathit{call}(\text{C1, P.finish})| & (6)\\
|\mathit{call}(\text{C1, P.finish})| &= |\mathit{resume}(\text{C1, P.start})| & (7)\\
|\mathit{resume}(\text{C1, P.start})| &= |\mathit{call}(\text{C1, P.start})|. & (8)
\end{aligned}$$

Since the projection of s on the alphabet of τ(P) lies in the language of

(P-1)*P-4, we have

$$\begin{aligned}
|\mathit{starve}_a(\mathrm{P.finish})| &= 1 && (9)\\
|\mathit{end\_rend}(\mathrm{C1, P.start})| &= |\mathit{beg\_rend}(\mathrm{C1, P.start})| && (10)\\
|\mathit{end\_rend}(\mathrm{C2, P.start})| &= |\mathit{beg\_rend}(\mathrm{C2, P.start})| && (11)\\
\sum_i |\mathit{beg\_rend}(\mathrm{Ci, P.start})| &= |\mathit{end\_rend}(\mathrm{O, P.act})| && (12)\\
|\mathit{end\_rend}(\mathrm{O, P.act})| &= |\mathit{beg\_rend}(\mathrm{O, P.act})| && (13)\\
|\mathit{beg\_rend}(\mathrm{O, P.act})| &= |\mathit{resume}(\mathrm{P, O.charge})| + 1 && (14)\\
|\mathit{resume}(\mathrm{P, O.charge})| &= |\mathit{call}(\mathrm{P, O.charge})| && (15)\\
|\mathit{call}(\mathrm{P, O.charge})| &= \sum_i |\mathit{end\_rend}(\mathrm{Ci, P.finish})| && (16)\\
|\mathit{end\_rend}(\mathrm{C1, P.finish})| &= |\mathit{beg\_rend}(\mathrm{C1, P.finish})| && (17)\\
|\mathit{end\_rend}(\mathrm{C2, P.finish})| &= |\mathit{beg\_rend}(\mathrm{C2, P.finish})|. && (18)
\end{aligned}$$

From the constraints of the form (5) of [3], we have

$$\begin{aligned}
|\mathit{call}(\mathrm{C1, O.prepay})| &= |\mathit{beg\_rend}(\mathrm{C1, O.prepay})| && (19)\\
|\mathit{call}(\mathrm{C1, P.start})| &= |\mathit{beg\_rend}(\mathrm{C1, P.start})| && (20)\\
|\mathit{call}(\mathrm{C1, P.finish})| &= |\mathit{beg\_rend}(\mathrm{C1, P.finish})| && (21)\\
|\mathit{call}(\mathrm{O, C1.change})| &= |\mathit{beg\_rend}(\mathrm{O, C1.change})| && (22)\\
|\mathit{call}(\mathrm{O, P.act})| &= |\mathit{beg\_rend}(\mathrm{O, P.act})| && (23)\\
|\mathit{call}(\mathrm{C2, P.start})| &= |\mathit{beg\_rend}(\mathrm{C2, P.start})| && (24)\\
|\mathit{call}(\mathrm{C2, P.finish})| &= |\mathit{beg\_rend}(\mathrm{C2, P.finish})| && (25)\\
|\mathit{call}(\mathrm{P, O.charge})| &= |\mathit{beg\_rend}(\mathrm{P, O.charge})|, && (26)
\end{aligned}$$

and, from the projection of s on the alphabet of $\tau(\mathrm{C2})$, we obtain

$$\begin{aligned}
|\mathit{beg\_rend}(\mathrm{O, C2.change})| &= |\mathit{resume}(\mathrm{C2, P.finish})| - |\mathrm{C2\text{-}6}| && (27)\\
|\mathit{resume}(\mathrm{C2, P.finish})| &= |\mathit{call}(\mathrm{C2, P.finish})| && (28)\\
|\mathit{call}(\mathrm{C2, P.finish})| &= |\mathit{resume}(\mathrm{C2, P.start})| - |\mathrm{C2\text{-}5}| && (29)\\
|\mathit{resume}(\mathrm{C2, P.start})| &= |\mathit{call}(\mathrm{C2, P.start})| && (30)\\
|\mathit{call}(\mathrm{C2, P.start})| &= |\mathit{resume}(\mathrm{C2, O.prepay})| - |\mathrm{C2\text{-}4}| && (31)\\
|\mathit{resume}(\mathrm{C2, O.prepay})| &= |\mathit{call}(\mathrm{C2, O.prepay})| - |\mathrm{C2\text{-}3}| && (32)\\
|\mathit{call}(\mathrm{C2, O.prepay})| &= |\mathit{end\_rend}(\mathrm{O, C2.change})| + 1 - |\mathrm{C2\text{-}2}| && (33)\\
|\mathit{end\_rend}(\mathrm{O, C2.change})| &= |\mathit{beg\_rend}(\mathrm{O, C2.change})| && (34)\\
|\mathit{starve}_c(\mathrm{C2, O.prepay})| &= |\mathrm{C2\text{-}2}| && (35)\\
|\mathit{dead\_rend}(\mathrm{C2, O.prepay})| &= |\mathrm{C2\text{-}3}| && (36)\\
|\mathit{starve}_c(\mathrm{C2, P.start})| &= |\mathrm{C2\text{-}4}| && (37)
\end{aligned}$$

$$|\mathit{starve}_c(\mathrm{C2}, \mathrm{P.finish})| = |\mathrm{C2\text{-}5}| \tag{38}$$
$$|\mathit{starve}_a(\mathrm{C2.change})| = |\mathrm{C2\text{-}6}| \tag{39}$$
$$|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| \geq |\mathrm{C2\text{-}3}| \tag{40}$$
$$|\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| \geq |\mathrm{C2\text{-}4}| \tag{41}$$
$$|\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| \geq |\mathrm{C2\text{-}5}| \tag{42}$$
$$|\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| \geq |\mathrm{C2\text{-}6}|, \tag{43}$$

and

$$|\mathrm{C2\text{-}2}| + |\mathrm{C2\text{-}3}| + |\mathrm{C2\text{-}4}| + |\mathrm{C2\text{-}5}| + |\mathrm{C2\text{-}6}| = 1. \tag{44}$$

Finally, a constraint of the form (11) of [3] gives

$$|\mathit{starve}_a(\mathrm{P.finish})| + |\mathit{starve}_c(\mathrm{C1}, \mathrm{P.finish})| \leq 1 \tag{45}$$
$$|\mathit{starve}_a(\mathrm{P.finish})| + |\mathit{starve}_c(\mathrm{C2}, \mathrm{P.finish})| \leq 1. \tag{46}$$

The integer linear programming package reports that this system is inconsistent.

3.2 Analysis of the Case in Which the PUMP Task Starves Calling the Entry OPERATOR.CHARGE

Now assume that we have a constrained prefix containing a $\mathit{starve}_c(\mathrm{C1}, \mathrm{P.start})$ symbol whose projection onto the alphabet of $\tau(\mathrm{P})$ lies in the language of the expression $(\mathrm{P\text{-}1})^*\mathrm{P\text{-}5}$.

As before, projecting on the alphabet of $\tau(\mathrm{C1})$ gives

$$|\mathit{starve}_c(\mathrm{C1}, \mathrm{P.start})| = 1 \tag{1}$$
$$|\mathit{resume}(\mathrm{C1}, \mathrm{O.prepay})| = |\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| \tag{2}$$
$$|\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| = |\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| + 1 \tag{3}$$
$$|\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| = |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| \tag{4}$$
$$|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| = |\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| \tag{5}$$
$$|\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| = |\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| \tag{6}$$
$$|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| = |\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| \tag{7}$$
$$|\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| = |\mathit{call}(\mathrm{C1}, \mathrm{P.start})|. \tag{8}$$

Our hypothesis about the projection of s on the alphabet of $\tau(\mathrm{P})$ implies that

$$|\mathit{starve}_c(\mathrm{P}, \mathrm{O.charge})| = 1 \tag{9}$$
$$|\mathit{end\_rend}(\mathrm{C1}, \mathrm{P.finish})| = |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.finish})| \tag{10}$$
$$|\mathit{end\_rend}(\mathrm{C2}, \mathrm{P.finish})| = |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.finish})| \tag{11}$$

$$
\begin{aligned}
\sum_i |\mathit{beg\_rend}(\text{Ci}, \text{P.finish})| &= \sum_i |\mathit{end\_rend}(\text{Ci}, \text{P.start})| && (12)\\
|\mathit{end\_rend}(\text{C1}, \text{P.start})| &= |\mathit{beg\_rend}(\text{C1}, \text{P.start})| && (13)\\
|\mathit{end\_rend}(\text{C2}, \text{P.start})| &= |\mathit{beg\_rend}(\text{C2}, \text{P.start})| && (14)\\
\sum_i |\mathit{beg\_rend}(\text{Ci}, \text{P.start})| &= |\mathit{end\_rend}(\text{O}, \text{P.act})| && (15)\\
|\mathit{end\_rend}(\text{O}, \text{P.act})| &= |\mathit{beg\_rend}(\text{O}, \text{P.act})| && (16)\\
|\mathit{beg\_rend}(\text{O}, \text{P.act})| &= |\mathit{resume}(\text{P}, \text{O.charge})| + 1 && (17)\\
|\mathit{resume}(\text{P}, \text{O.charge})| &= |\mathit{call}(\text{P}, \text{O.charge})| && (18)\\
\sum_i |\mathit{end\_rend}(\text{Ci}, \text{P.finish})| &\geq |\mathit{starve}_c(\text{P}, \text{O.charge})|. && (19)
\end{aligned}
$$

Exactly as before, we also have

$$
\begin{aligned}
|\mathit{call}(\text{C1}, \text{O.prepay})| &= |\mathit{beg\_rend}(\text{C1}, \text{O.prepay})| && (20)\\
|\mathit{call}(\text{C1}, \text{P.start})| &= |\mathit{beg\_rend}(\text{C1}, \text{P.start})| && (21)\\
|\mathit{call}(\text{C1}, \text{P.finish})| &= |\mathit{beg\_rend}(\text{C1}, \text{P.finish})| && (22)\\
|\mathit{call}(\text{O}, \text{C1.change})| &= |\mathit{beg\_rend}(\text{O}, \text{C1.change})| && (23)\\
|\mathit{call}(\text{O}, \text{P.act})| &= |\mathit{beg\_rend}(\text{O}, \text{P.act})| && (24)\\
|\mathit{call}(\text{C2}, \text{P.start})| &= |\mathit{beg\_rend}(\text{C2}, \text{P.start})| && (25)\\
|\mathit{call}(\text{C2}, \text{P.finish})| &= |\mathit{beg\_rend}(\text{C2}, \text{P.finish})| && (26)\\
|\mathit{call}(\text{P}, \text{O.charge})| &= |\mathit{beg\_rend}(\text{P}, \text{O.charge})| && (27)\\
|\mathit{beg\_rend}(\text{O}, \text{C2.change})| &= |\mathit{resume}(\text{C2}, \text{P.finish})| - |\text{C2-6}| && (28)\\
|\mathit{resume}(\text{C2}, \text{P.finish})| &= |\mathit{call}(\text{C2}, \text{P.finish})| && (29)\\
|\mathit{call}(\text{C2}, \text{P.finish})| &= |\mathit{resume}(\text{C2}, \text{P.start})| - |\text{C2-5}| && (30)\\
|\mathit{resume}(\text{C2}, \text{P.start})| &= |\mathit{call}(\text{C2}, \text{P.start})| && (31)\\
|\mathit{call}(\text{C2}, \text{P.start})| &= |\mathit{resume}(\text{C2}, \text{O.prepay})| - |\text{C2-4}| && (32)\\
|\mathit{resume}(\text{C2}, \text{O.prepay})| &= |\mathit{call}(\text{C2}, \text{O.prepay})| - |\text{C2-3}| && (33)\\
|\mathit{call}(\text{C2}, \text{O.prepay})| &= |\mathit{end\_rend}(\text{O}, \text{C2.change})| + 1 - |\text{C2-2}| && (34)\\
|\mathit{end\_rend}(\text{O}, \text{C2.change})| &= |\mathit{beg\_rend}(\text{O}, \text{C2.change})| && (35)\\
|\mathit{starve}_c(\text{C2}, \text{O.prepay})| &= |\text{C2-2}| && (36)\\
|\mathit{dead\_rend}(\text{C2}, \text{O.prepay})| &= |\text{C2-3}| && (37)\\
|\mathit{starve}_c(\text{C2}, \text{P.start})| &= |\text{C2-4}| && (38)\\
|\mathit{starve}_c(\text{C2}, \text{P.finish})| &= |\text{C2-5}| && (39)\\
|\mathit{starve}_a(\text{C2.change})| &= |\text{C2-6}| && (40)\\
|\mathit{call}(\text{C2}, \text{O.prepay})| &\geq |\text{C2-3}| && (41)\\
|\mathit{resume}(\text{C2}, \text{O.prepay})| &\geq |\text{C2-4}| && (42)\\
|\mathit{resume}(\text{C2}, \text{P.start})| &> |\text{C2-5}| && (43)
\end{aligned}
$$

$$|resume(\text{C2}, \text{P.finish})| \geq |\text{C2-6}|, \tag{44}$$

and

$$|\text{C2-2}| + |\text{C2-3}| + |\text{C2-4}| + |\text{C2-5}| + |\text{C2-6}| = 1. \tag{45}$$

From constraints of the form (5) of [3], we have

$$|call(\text{C2}, \text{O.prepay})| = |beg\_rend(\text{C2}, \text{O.prepay})| \tag{46}$$
$$|call(\text{O}, \text{C2.change})| = |beg\_rend(\text{O}, \text{C2.change})|. \tag{47}$$

From constraints of the form (11) of [3], we have

$$|starve_a(\text{P.finish})| + |starve_c(\text{C2}, \text{P.finish})| \leq 1 \tag{48}$$
$$|starve_c(\text{P}, \text{O.charge})| + |starve_a(\text{O.charge})| \leq 1. \tag{49}$$

As before, we now examine the projection of s on the alphabet of $\tau(\text{O})$. By (9) and (45), we have $|starve_a(\text{O.charge})| = 0$, so the projection must lie in the language of (O-1 $\cdots$ O-11)*(O-13 $\vee$ (O-14 O-15) $\vee$ (O-16(O-17 $\vee$ O-18)).

3.2.1 The (O-1 $\cdots$ O-11)*O-13 Case

Assume that the projection of s on the alphabet of $\tau(\text{O})$ lies in the language of (O-1 $\cdots$ O-11)*O-13. Then s represents a behavior in which the OPERATOR task starves calling the entry PUMP.ACTIVATE. We have

$$|kill\_rend(\text{O.prepay})| = 1 \tag{50}$$
$$|starve_c(\text{O}, \text{P.act})| = 1 \tag{51}$$
$$\sum_i |beg\_rend(\text{Ci}, \text{O.prepay})| = \sum_i |end\_rend(\text{Ci}, \text{O.prepay})| + 1 \tag{52}$$
$$|resume(\text{O}, \text{C2.change})| = |call(\text{O}, \text{C2.change})| \tag{53}$$
$$|resume(\text{O}, \text{C1.change})| = |call(\text{O}, \text{C1.change})| \tag{54}$$
$$\sum_i |call(\text{O}, \text{Ci.change})| = |end\_rend(\text{P}, \text{O.charge})| \tag{55}$$
$$|end\_rend(\text{P}, \text{O.charge})| = |beg\_rend(\text{P}, \text{O.charge})|. \tag{56}$$

From the *kill_rend-dead_rend* constraints, it follows that

$$|kill\_rend(\text{O.prepay})| - \sum_i |dead\_rend(\text{Ci}, \text{O.prepay})| = 0, \tag{57}$$

and we know that

$$|dead\_rend(\text{C1}, \text{O.prepay}) = 0. \tag{58}$$

From constraints of the form (5), we have

$$|end\_rend(\text{C1}, \text{O.prepay})| = |resume(\text{C1}, \text{O.prepay})| \tag{59}$$
$$|end\_rend(\text{C2}, \text{O.prepay})| = |resume(\text{C2}, \text{O.prepay})|. \tag{60}$$

The integer linear programming package reports that this system is inconsistent.

3.2.2 The $(\text{O-1}\cdots\text{O-11})^*\text{O-14 O-15}$ Case

Assume that the projection of the constrained prefix s on the alphabet of $\tau(\text{O})$ lies in the language of the expression $(\text{O-1}\cdots\text{O-11})^*\text{O-14 O-15}$. Then s represents a behavior in which the OPERATOR task starves calling the entry PUMP.ACTIVATE following a rendezvous at the entry OPERATOR.CHARGE. We have

$$\begin{aligned}
|\mathit{starve}_c(\text{O}, \text{P.act})| &= 1 && (50)\\
|\mathit{end\_rend}(\text{P}, \text{O.charge})| &\geq 1 && (51)\\
|\mathit{beg\_rend}(\text{P}, \text{O.charge})| &= |\mathit{end\_rend}(\text{P}, \text{O.charge})| && (52)\\
\sum_i |\mathit{resume}(\text{O}, \text{Ci.change})| &= |\mathit{beg\_rend}(\text{P}, \text{O.charge})| - 1 && (53)\\
|\mathit{call}(\text{O}, \text{C2.change})| &= |\mathit{resume}(\text{O}, \text{C2.change})| && (54)\\
|\mathit{call}(\text{O}, \text{C1.change})| &= |\mathit{resume}(\text{O}, \text{C1.change})|. && (55)
\end{aligned}$$

The integer linear programming package reports that this system is inconsistent.

3.2.3 The $(\text{O-1}\cdots\text{O-11})^*\text{O-16}(\text{O-17} \vee \text{O-18})$ Case

Assume that the projection of s on the alphabet to $\tau(\text{O})$ lies in the language of the expression $(\text{O-1}\cdots\text{O-11})^*\text{O-16}(\text{O-17} \vee \text{O-18})$. Then s represents a behavior in which the OPERATOR task starves calling one of the CUSTOMER_i.CHANGE entries. We have

$$\begin{aligned}
|\mathit{starve}_c(\text{O}, \text{C1.change})| &= |\text{O-17}| && (50)\\
|\mathit{starve}_c(\text{O}, \text{C2.change})| &= |\text{O-18}| && (51)\\
|\text{O-17}| + |\text{O-18}| &= 1 && (52)\\
|\mathit{call}(\text{O}, \text{P.act})| &= |\mathit{resume}(\text{O}, \text{P.act})| && (53)\\
|\mathit{beg\_rend}(\text{P}, \text{O.charge})| &= |\mathit{end\_rend}(\text{P}, \text{O.charge})| && (54)\\
\sum_i |\mathit{resume}(\text{O}, \text{Ci.change})| &= |\mathit{beg\_rend}(\text{P}, \text{O.charge})| - 1 && (55)\\
|\mathit{call}(\text{O}, \text{C2.change})| &= |\mathit{resume}(\text{O}, \text{C2.change})| && (56)\\
|\mathit{call}(\text{O}, \text{C1.change})| &= |\mathit{resume}(\text{O}, \text{C1.change})|. && (57)
\end{aligned}$$

The integer linear programming package reports that this system is inconsistent.

3.3 Analysis of the Case in Which the PUMP Task Starves at the Entry PUMP.ACTIVATE

Assume that s is a constrained prefix containing a $\mathit{starve}_c(\text{C1}, \text{P.start})$ symbol and that the projection of s onto the alphabet of $\tau(\text{P})$ lies in the language of the expression $(\text{P-1})^*\text{P-2}$.

As before, projecting on the alphabet of $\tau(\mathrm{C1})$ gives

$$\begin{aligned}
|\mathit{starve}_c(\mathrm{C1}, \mathrm{P.start})| &= 1 && (1)\\
|\mathit{resume}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| && (2)\\
|\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| + 1 && (3)\\
|\mathit{end\_rend}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| && (4)\\
|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| && (5)\\
|\mathit{resume}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| && (6)\\
|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| && (7)\\
|\mathit{resume}(\mathrm{C1}, \mathrm{P.start})| &= |\mathit{call}(\mathrm{C1}, \mathrm{P.start})|. && (8)
\end{aligned}$$

Our hypothesis about the projection of s on the alphabet of $\tau(\mathrm{P})$ implies that

$$\begin{aligned}
|\mathit{starve}_a(\mathrm{P.act})| &= 1 && (9)\\
|\mathit{call}(\mathrm{P}, \mathrm{O.charge}) &= |\mathit{resume}(\mathrm{P}, \mathrm{O.charge})| && (10)\\
\sum_i |\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.finish})| &= |\mathit{call}(\mathrm{P}, \mathrm{O.charge})| && (11)\\
|\mathit{end\_rend}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.finish})| && (12)\\
|\mathit{end\_rend}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.finish})| && (13)\\
\sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.finish})| &= \sum_i |\mathit{end\_rend}(\mathrm{Ci}, \mathrm{P.start})| && (14)\\
|\mathit{end\_rend}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.start})| && (15)\\
|\mathit{end\_rend}(\mathrm{C1}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.start})| && (16)\\
\sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{P.start})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})| && (17)\\
|\mathit{end\_rend}(\mathrm{O}, \mathrm{P.act})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})|. && (18)
\end{aligned}$$

Exactly as before, we also have

$$\begin{aligned}
|\mathit{call}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{O.prepay})| && (19)\\
|\mathit{call}(\mathrm{C1}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.start})| && (20)\\
|\mathit{call}(\mathrm{C1}, \mathrm{P.finish})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{P.finish})| && (21)\\
|\mathit{call}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C1.change})| && (22)\\
|\mathit{call}(\mathrm{O}, \mathrm{P.act})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})| && (23)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.start})| && (24)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{P.finish})| && (25)\\
|\mathit{call}(\mathrm{P}, \mathrm{O.charge})| &= |\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})| && (26)\\
|\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| - |\mathrm{C2\text{-}6}| && (27)
\end{aligned}$$

$$
\begin{aligned}
|\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| && (28)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.finish})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| - |\text{C2-5}| && (29)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{call}(\mathrm{C2}, \mathrm{P.start})| && (30)\\
|\mathit{call}(\mathrm{C2}, \mathrm{P.start})| &= |\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| - |\text{C2-4}| && (31)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| - |\text{C2-3}| && (32)\\
|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{end\_rend}(\mathrm{O}, \mathrm{C2.change})| + 1 - |\text{C2-2}| && (33)\\
|\mathit{end\_rend}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| && (34)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{O.prepay})| &= |\text{C2-2}| && (35)\\
|\mathit{dead\_rend}(\mathrm{C2}, \mathrm{O.prepay})| &= |\text{C2-3}| && (36)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{P.start})| &= |\text{C2-4}| && (37)\\
|\mathit{starve}_c(\mathrm{C2}, \mathrm{P.finish})| &= |\text{C2-5}| && (38)\\
|\mathit{starve}_a(\mathrm{C2.change})| &= |\text{C2-6}| && (39)\\
|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| &\geq |\text{C2-3}| && (40)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{O.prepay})| &\geq |\text{C2-4}| && (41)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.start})| &\geq |\text{C2-5}| && (42)\\
|\mathit{resume}(\mathrm{C2}, \mathrm{P.finish})| &\geq |\text{C2-6}|, && (43)
\end{aligned}
$$

and

$$|\text{C2-2}| + |\text{C2-3}| + |\text{C2-4}| + |\text{C2-5}| + |\text{C2-6}| = 1. \qquad (44)$$

Constraints of the forms (5) and (11) of [3] yield

$$
\begin{aligned}
|\mathit{call}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{O.prepay})| && (45)\\
|\mathit{call}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{beg\_rend}(\mathrm{O}, \mathrm{C2.change})| && (46)\\
|\mathit{starve}_a(\mathrm{P.act})| + |\mathit{starve}_c(\mathrm{O}, \mathrm{P.act})| &\leq 1. && (47)
\end{aligned}
$$

We now have to examine the projection of s on the alphabet of $\tau(\mathrm{O})$. By (9) and (47), we have $|\mathit{starve}_c(\mathrm{O}, \mathrm{P.act})| = 0$, so the projection must lie in the language of the expression $(\text{O-1} \cdots \text{O-11})^*(\text{O-12} \vee \text{O-16}(\text{O-17} \vee \text{O-18}))$. We consider these cases separately.

3.3.1 The $(\text{O-1} \cdots \text{O-11})^*\text{O-12}$ Case

Assume that the projection of s on the alphabet of $\tau(\mathrm{O})$ lies in the language of the expression $(\text{O-1} \cdots \text{O-11})^*\text{O-12}$. Then s represents a behavior in which the task **OPERATOR** starves waiting for a call at either of the entries **OPERATOR.PREPAY** or **OPERATOR.CHARGE**. We have

$$
\begin{aligned}
|\mathit{starve}_a(\mathrm{O.prepay})| &= 1 && (48)\\
|\mathit{starve}_a(\mathrm{O.charge})| &= 1 && (49)\\
|\mathit{resume}(\mathrm{O}, \mathrm{C2.change})| & \quad |\mathit{call}(\mathrm{O}, \mathrm{C2.change})| && (50)
\end{aligned}
$$

$$\begin{aligned}
|\mathit{resume}(\text{O}, \text{C1.change})| &= |\mathit{call}(\text{O}, \text{C1.change})| && (51)\\
\sum_i |\mathit{call}(\text{O}, \text{Ci.change})| &= |\mathit{end\_rend}(\text{P}, \text{O.charge})| && (52)\\
|\mathit{end\_rend}(\text{P}, \text{O.charge})| &= |\mathit{beg\_rend}(\text{P}, \text{O.charge})| && (53)\\
|\mathit{end\_rend}(\text{C1}, \text{O.prepay})| &= |\mathit{beg\_rend}(\text{C1}, \text{O.prepay})| && (54)\\
|\mathit{end\_rend}(\text{C2}, \text{O.prepay})| &= |\mathit{beg\_rend}(\text{C2}, \text{O.prepay})| && (55)\\
|\mathit{resume}(\text{O}, \text{P.act})| &= |\mathit{call}(\text{O}, \text{P.act}) && (56)\\
|\mathit{kill\_rend}(\text{O.prepay})| &= 0. && (57)
\end{aligned}$$

Constraints of the form (11) give

$$\begin{aligned}
|\mathit{starve}_c(\text{C1}, \text{O.prepay})| + |\mathit{starve}_a(\text{O.prepay})| &\leq 1 && (58)\\
|\mathit{starve}_c(\text{C2}, \text{O.prepay})| + |\mathit{starve}_a(\text{O.prepay})| &\leq 1, && (59)
\end{aligned}$$

and we our hypothesis on s implies that

$$|\mathit{starve}_c(\text{C1}, \text{O.prepay})| = 0. \qquad (60)$$

Finally, we have

$$\begin{aligned}
|\mathit{kill\_rend}(\text{O.prepay})| - \sum_i |\mathit{dead\_rend}(\text{Ci}, \text{O.prepay})| &= 0 && (61)\\
|\mathit{dead\_rend}(\text{C1}, \text{O.prepay})| &= 0. && (62)
\end{aligned}$$

The integer linear programming package finds a solution to this system. That solution represents a behavior in which each customer task completes a rendezvous at the entry OPERATOR.PREPAY, but the operator task never calls the entry PUMP.ACTIVATE. By the argument for the lemma in section 2.5.1, modified slightly to account for the fact that we have changed the order of the calls to the CHANGE and ACTIVATE entries in the accept CHARGE alternative, we have

$$\sum_i |\mathit{resume}(\text{Ci}, \text{O.prepay})| \leq |\mathit{resume}(\text{O}, \text{P.act})| + 1. \qquad (63)$$

This last inequality makes the system inconsistent.

3.3.2 The (O-1 ⋯ O-11)* O-16(O-17 ∨ O-18) Case

Assume that the projection of s on the alphabet of $\tau(\text{O})$ lies in the language of the expression (O-1 ⋯ O-11)*O-16(O-17 ∨ O-18). Then

$$\begin{aligned}
|\mathit{starve}_c(\text{O}, \text{C1.change})| &= |\text{O-17}| && (48)\\
|\mathit{starve}_c(\text{O}, \text{C2.change})| &= |\text{O-18}| && (49)\\
|\text{O-17}| + |\text{O-18}| &= 1 && (50)
\end{aligned}$$

$$
\begin{aligned}
|\mathit{resume}(\mathrm{O}, \mathrm{C1.change})| &= |\mathit{call}(\mathrm{O}, \mathrm{C1.change})| && (51)\\
|\mathit{resume}(\mathrm{O}, \mathrm{C2.change})| &= |\mathit{call}(\mathrm{O}, \mathrm{C2.change})| && (52)\\
|\mathit{resume}(\mathrm{O}, \mathrm{P.act}) &= |\mathit{call}(\mathrm{O}, \mathrm{P.act})| && (53)\\
|\mathit{end\_rend}(\mathrm{P}, \mathrm{O.charge})| &\geq 1 && (54)\\
|\mathit{end\_rend}(\mathrm{P}, \mathrm{O.charge})| &= |\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})| && (55)\\
|\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})| &= \sum_i |\mathit{call}(\mathrm{O}, \mathrm{Ci.change})| + 1 && (56)\\
|\mathit{end\_rend}(\mathrm{C1}, \mathrm{O.prepay})| &= |\mathit{beg\_rend}(\mathrm{C1}, \mathrm{O.prepay})| && (57)\\
|\mathit{end\_rend}(\mathrm{C2}, \mathrm{O.prepay})| &= |\mathit{beg\_rend}(\mathrm{C2}, \mathrm{O.prepay})| && (58)\\
|\mathit{kill\_rend}(\mathrm{O.prepay})| &= 0. && (59)
\end{aligned}
$$

From constraints of type (11), we have

$$
\begin{aligned}
|\mathit{starve}_a(\mathrm{C1.change})| + |\mathit{starve}_c(\mathrm{O}, \mathrm{C1.change})| &\leq 1 && (60)\\
|\mathit{starve}_a(\mathrm{C2.change})| + |\mathit{starve}_c(\mathrm{O}, \mathrm{C2.change})| &\leq 1, && (61)
\end{aligned}
$$

and our hypotheses on s tell us that

$$|\mathit{starve}_a(\mathrm{C1.change}) = 0. \qquad (62)$$

We also have

$$
\begin{aligned}
|\mathit{kill\_rend}(\mathrm{O.prepay})| - \sum_i |\mathit{dead\_rend}(\mathrm{Ci}, \mathrm{O.prepay})| &= 0 && (63)\\
|\mathit{dead\_rend}(\mathrm{C1}, \mathrm{O.prepay})| &= 0. && (64)
\end{aligned}
$$

The integer linear programming package reports a solution for this system. The solution corresponds to a behavior in which both customer tasks prepay, the operator activates the pump, CUSTOMER_2 pumps gas and starves waiting for change, and the operator starves trying to give change to CUSTOMER_1. Examining the CEDL code, we see that such a behavior is impossible. The value of the variable cus following the rendezvous at the entry OPERATOR.CHARGE must be one or two, depending on whether the quantity

$$\sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{O.prepay})| - \sum_i |\mathit{resume}(\mathrm{O}, \mathrm{Ci.change})|$$

is 1 or 2 (it's clear that it can't be 0). In the first case, no $\mathit{call}(\mathrm{O}, \mathrm{P.act})$ symbol can occur after the last $\mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})$ symbol, while in the second case, one must occur. Using (23), (26), and the fact that $\mathit{beg\_rend}(\mathrm{O}, \mathrm{P.act})$ and $\mathit{call}(\mathrm{P}, \mathrm{O.charge})$ symbols alternate in $\tau(\mathrm{P})$ as we did in secton 2.4.3, we have

$$
\begin{aligned}
|\mathit{call}(\mathrm{O}, \mathrm{P.act})| &= |\mathit{call}(\mathrm{P}, \mathrm{O.charge})| + \sum_i |\mathit{beg\_rend}(\mathrm{Ci}, \mathrm{O.prepay})|\\
&\quad - \sum_i |\mathit{resume}(\mathrm{O}, \mathrm{Ci.change})| - 1. && (65)
\end{aligned}
$$

Adding this equation makes the system inconsistent, and we conclude that no such constrained prefix exists.

4 A Three-Customer System

The systems discussed in this report are based on the examples given by Helmbold and Luckham in [7]. To facilitate the analysis of these systems, we reduced the number of tasks, cutting the number of customer tasks from ten to two and the number of pump tasks from three to one. In this section, we briefly examine the effect of this reduction on the analysis by considering a system with three customers.

4.1 The Task Expressions and the Behavior of the System

In Figures 13 and 14, we show the task declarations and the body of the `OPERATOR` task for a three-customer version of the revised gas-station system considered in section 3. The task expression for the `OPERATOR` task is given in Figure 15. The other task bodies and task expressions are the same as in section 3.

We will not present here a detailed analysis of this larger system. We simply note that this system does have behaviors in which a customer prepays but never pumps gas. A constrained prefix representing such a behavior is

beg_loop(C1)*beg_loop*(P)*beg_loop*(C2)*beg_loop*(C3)*def*(cus, zero)
def(current, $\perp$)*def*(wait, $\perp$)*beg_loop*(O)*call*(C1, O.prepay)
beg_rend(C1, O.prepay)*def*(cus_id, c1)*use*(cus, zero)*def*(cus, one)*use*(cus, one)
def(current, c1)*call*(O, P.act)*beg_rend*(O, P.act)*end_rend*(O, P.act)
resume(O, P.act)*end_rend*(C1, O.prepay)*resume*(C1, O.prepay)
call(C2, O.prepay)*beg_rend*(C2, O.prepay)*def*(cus_id, c2)*use*(cus, one)
def(cus, two)*use*(cus, two)*def*(wait_1, c2)*end_rend*(C2, O.prepay)
resume(C2, O.prepay)*call*(C2, P.start)*beg_rend*(C2, P.start)
end_rend(C2, P.start)*resume*(C2, P.start)*call*(C2, P.finish)
beg_rend(C2, P.finish)*end_rend*(C2, P.finish)*resume*(C2, P.finish)
call(C2, O.prepay)*beg_rend*(C2, O.prepay)*def*(cus_id, c3)*use*(cus, two)
def(cus, three)*use*(cus, three)*def*(wait_2, c3)*end_rend*(C3, O.prepay)
resume(C3, O.prepay)*call*(P, O.charge)*beg_rend*(P, O.charge)
end_rend(P, O.charge)*resume*(P, O.charge)*use*(cus, three)*call*(O, P.act)
beg_rend(O, P.act)*end_rend*(O, P.act)*resume*(O, P.act)*call*(C3, P.start)
end_rend(C3, P.start)*resume*(C3, P.start)*call*(C3, P.finish)
beg_rend(C3, P.finish)*end_rend*(C3, P.finish)*resume*(C3, P.finish)

```
package COMMON is
  type C_NAME is (c1,c2,c3);  -- enough for three customers
  type COUNTER is (zero,one,two,three);
                           -- enough to handle 3 customers
end COMMON;

use COMMON;
task OPERATOR is
  entry PREPAY(CUSTOMER_ID : in C_NAME);
  entry CHARGE;
end OPERATOR;

task PUMP is
  entry ACTIVATE;
  entry START_PUMPING;
  entry FINISH_PUMPING;
end PUMP;

use COMMON;
task CUSTOMER_1 is
  entry CHANGE;
end CUSTOMER_1;

use COMMON;
task CUSTOMER_2 is
  entry CHANGE;
end CUSTOMER_2;

use COMMON;
task CUSTOMER_3 is
  entry CHANGE;
end CUSTOMER_3;
```

Figure 13: Task declarations for the three-customer system

```
use COMMON;
task body OPERATOR is
  CUSTOMERS : COUNTER := zero;
  CURRENT, WAITING_1, WAITING_2 : C_NAME;
  begin
    loop
      select
        accept PREPAY(CUSTOMER_ID : in C_NAME) do
          CUSTOMERS := COUNTER'succ(CUSTOMERS);
          if CUSTOMERS = one then     -- if no previous customer
                                      -- is waiting
            CURRENT := CUSTOMER_ID;   -- mark this one as current
            PUMP.ACTIVATE;            -- and activate the pump
          elsif CUSTOMERS = two then    -- this is the second one
            WAITING_1 := CUSTOMER_ID;    -- mark it as the second
                                         -- in line
          else
            WAITING_2 := CUSTOMER_ID;    -- otherwise, mark this
                                         -- one as third in line
          end if;
        end PREPAY;
      or
        accept CHARGE;
        if CUSTOMERS > one then          -- if another customer is
                                         -- waiting,
          PUMP.ACTIVATE;                 -- activate the pump
        end if;
        if CURRENT = c1 then
          CUSTOMER_1.CHANGE;
        elsif CURRENT = c2 then
          CUSTOMER_2.CHANGE;
        else
          CUSTOMER_3.CHANGE;
        end if;
        CUSTOMERS := COUNTER'pred(CUSTOMERS);
```

Figure 14: Body of the OPERATOR Task for the Three-Customer System

```
        if CUSTOMERS = one then     -- if one customer is
          CURRENT := WAITING_1;     -- waiting, promote that one
                                    -- to be current
        elsif CUSTOMERS = two then  -- if two customers waiting
          CURRENT := WAITING_1;      -- promote both of them
          WAITING_1 := WAITING_2;
        end if;
      end select;
    end loop;
  end OPERATOR;
```

Figure 14: Continued

$$\mathit{starve}_c(\mathrm{P}, \mathrm{O.charge})\mathit{use}(\mathrm{current}, \mathrm{c1})\mathit{starve}_c(\mathrm{O}, \mathrm{C1.change})$$
$$\mathit{starve}_c(\mathrm{C1}, \mathrm{P.start})\mathit{starve}_a(\mathrm{C2.change})\mathit{starve}_a(\mathrm{C3.change}).$$

In terms of the gas station modelled by the CEDL system, this prefix corresponds to a situation in which the first customer prepays, the operator activates the pump and a second customer prepays and pumps gas. A third customer then prepays and the pump indicates to the operator that a customer has finished. Knowing that customers are waiting, the operator activates the pump again and the third customer pumps gas. The operator then tries to give change to the first customer, who is still waiting to pump, and the other two customers wait for change.

Our analysis of the two-customer system, then, does not carry over to the three-customer version. In our revised two-customer system, the operator reactivates the pump immediately when there is a customer waiting and the pump reports that a customer has finished pumping. When there are only two customers, this is enough to ensure that no customer prepays without getting to pump gas. In the three-customer version, however, the third customer can take advantage of this reactivation of the pump, preventing the first customer from pumping.

The important point is that, although this problem is fairly obvious once it has been pointed out, examination of the two-customer system does not clearly suggest that an enlarged version will behave in a radically different manner. With other systems, the combinatorics may be significantly more complicated, and there seems to be no good general approach to determining whether scaling the system down to facilitate analysis will actually invalidate the analysis. For this reason, it is especially important to try to construct automated tools that can handle systems of realistic sizes.

O-1 $\mathit{def}(\text{cus}, \text{zero})\mathit{def}(\text{current}, \bot)\mathit{def}(\text{wait}, \bot)\mathit{beg\_loop}(\text{O})\Big($

O-2 $\Big(\bigvee_{i}(\mathit{beg\_rend}(\text{Ci}, \text{O.prepay})\mathit{def}(\text{cus\_id}, \text{ci})(\bigvee_{x} \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{cus}, \text{succ}(\text{x})))$

O-3 $\Big((\mathit{use}(\text{cus}, \text{one})\mathit{def}(\text{current}, \text{ci})\mathit{call}(\text{O}, \text{P.act})\mathit{resume}(\text{O}, \text{P.act}))$

O-4 $\vee \mathit{use}(\text{cus}, \text{two})\mathit{def}(\text{wait\_1}, \text{ci}) \vee \mathit{use}(\text{cus}, \text{three})\mathit{def}(\text{wait\_2}, \text{ci})\Big)$

$\mathit{end\_rend}(\text{Ci}, \text{O.prepay}))\Big)$

O-5 $\vee\Big(\mathit{beg\_rend}(\text{P}, \text{O.charge})\mathit{end\_rend}(\text{P}, \text{O.charge})$

O-6 $((\bigvee_{\text{x>one}} \mathit{use}(\text{cus}, \text{x})\mathit{call}(\text{O}, \text{P.act})\mathit{resume}(\text{O}, \text{P.act})) \vee \bigvee_{\text{x}\leq\text{one}} \mathit{use}(\text{cus}, \text{x}))$

O-7 $(\mathit{use}(\text{current}, \text{c1})\mathit{call}(\text{O}, \text{C1.change})\mathit{resume}(\text{O}, \text{C1.change})$

O-8 $\vee \mathit{use}(\text{current}, \text{c2})\mathit{call}(\text{O}, \text{C2.change})\mathit{resume}(\text{O}, \text{C2.change}))$

O-8 $\vee \mathit{use}(\text{current}, \text{c3})\mathit{call}(\text{O}, \text{C3.change})\mathit{resume}(\text{O}, \text{C3.change}))$

O-10 $(\bigvee_{\text{x}} \mathit{use}(\text{cus}, \text{x})\mathit{def}(\text{cus}, \text{pred}(\text{x})))$

O-11 $(\mathit{use}(\text{cus}, \text{one})(\bigvee_{i} \mathit{use}(\text{wait\_1}, \text{ci})\mathit{def}(\text{current}, \text{ci}))$

$\mathit{use}(\text{cus}, \text{two})(\bigvee_{i} \mathit{use}(\text{wait\_1}, \text{ci})\mathit{def}(\text{current}, \text{ci}))$

O-12 $(\bigvee_{i} \mathit{use}(\text{wait\_2}, \text{ci})\mathit{def}(\text{wait\_1}, \text{ci})) \vee \mathit{use}(\text{cus}, \text{zero}))\Big)\Big)^{*}$

O-13 $\Big(\mathit{starve}_a(\text{O.prepay})\mathit{starve}_a(\text{O.charge})\mathit{stop}(\text{O})$

Figure 15: Task Expression $\tau(\text{O})$ Associated with the Three-Customer Version of the Task `OPERATOR`

O-14 $\vee(\bigvee_{i} \mathit{beg\_rend}(\mathrm{Ci}, \mathrm{O.prepay})\mathit{def}(\mathrm{cus\_id}, \mathrm{ci})(\bigvee_{x} \mathit{use}(\mathrm{cus}, \mathrm{x})$

$\mathit{def}(\mathrm{cus}, \mathrm{succ}(\mathrm{x})))\mathit{use}(\mathrm{cus}, \mathrm{one})\mathit{def}(\mathrm{current}, \mathrm{ci})\mathit{starve}_c(\mathrm{O}, \mathrm{P.act})$

$\mathit{kill\_rend}(\mathrm{O.prepay})\mathit{stop}(\mathrm{O}))$

O-15 $\vee \mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})\mathit{end\_rend}(\mathrm{P}, \mathrm{O.charge})$

O-16 $\Big((\bigvee_{x>\mathrm{one}} \mathit{use}(\mathrm{cus}, \mathrm{x}))\mathit{starve}_c(\mathrm{O}, \mathrm{P.act})\mathit{stop}(\mathrm{O})$

O-17 $\vee \mathit{beg\_rend}(\mathrm{P}, \mathrm{O.charge})\mathit{end\_rend}(\mathrm{P}, \mathrm{O.charge})((\bigvee_{x>\mathrm{one}} \mathit{use}(\mathrm{cus}, \mathrm{x}))$

$\mathit{call}(\mathrm{O}, \mathrm{P.act})\mathit{resume}(\mathrm{O}, \mathrm{P.act}) \vee \bigvee_{x\leq\mathrm{one}} \mathit{use}(\mathrm{cus}, \mathrm{x}))$

O-18 $\Big(\mathit{use}(\mathrm{current}, \mathrm{c1})\mathit{starve}_c(\mathrm{O}, \mathrm{C1.change})\mathit{stop}(\mathrm{O})$

O-19 $\vee \mathit{use}(\mathrm{current}, \mathrm{c2})\mathit{starve}_c(\mathrm{O}, \mathrm{C2.change})\mathit{stop}(\mathrm{O})$

O-20 $\vee \mathit{use}(\mathrm{current}, \mathrm{c3})\mathit{starve}_c(\mathrm{O}, \mathrm{C3.change})\mathit{stop}(\mathrm{O})\Big)\Big)$

Figure 15: (Continued)

5 Conclusion

In this report, we have presented parts of the analysis of three versions of a design for a concurrent software system. The analysis was based on constrained expression representations for the system, and involved the generation of systems of inequalities from those representations. The consistency or inconsistency of those inequalities reflects the existence or nonexistence of a behavior of the software system having some specified property.

Part of our purpose in carrying out this analysis was to investigate the prospects for automating it. For that reason, we did not attempt to generate inequalities in the cleverest or most insightful fashion, but rather tried to discover and follow a small number of simple rules. Most of these will be apparent to the reader, but we have not tried to give a full codification of them in this report. For those parts of the analysis where it was sufficient to consider only inequalities involving the numbers of occurrences of event symbols in an entire constrained prefix, this approach seems to be adequate. In those parts of the analysis involving data dependencies, where the constraints give information about segments of constrained prefixes, a more sophisticated approach was necessary. In such cases, we need to consider inequalities involving such things as the number of occurrences of a certain symbol following the last occurrence of another symbol.

While we have not yet found any general approach for such cases, we were encouraged to find that our more mechanical methods seemed to isolate and highlight the problems. For example, in section 2.5.1, the system of inequalities produced by our mechanical approach is consistent. However, examination of the solution found by the integer linear programming package shows that it cannot correspond to an actual behavior of the CEDL system. The solution corresponds to a situation in which the two `CUSTOMER` tasks both complete rendezvous at the entry `OPERATOR.PREPAY` but the `OPERATOR` task never calls the entry `PUMP.ACTIVATE`. Having had our attention focused on this problem, we can then establish another inequality relating the number of *resume*(Ci, O.prepay) symbols to the number of *resume*(O, P.act) symbols and reflecting the appropriate dependence on the value of the variable cus.

The advantages of this sort of analysis, based on the constrained expression formalism, include its rigor, the fact that it is exhaustive (unlike dynamic methods which animate the system in some way and must try to "execute" each possible behavior), and the focused nature of the analysis. Because this approach to analysis begins with a hypothesis about a behavior and reasons "backwards" from that, the problem of combinatorial explosion is reduced. It is clear, nonetheless, that automated support is necessary for this kind of analysis to be of much practical value. The size and complexity of the systems considered here are close to the limits of our ability to analyze them by hand. Even though these methods are intended to be used with relatively high-level designs, the importance of working with systems of realistic size should be clear from the discussion in the previous section. Even partial results on modularizing this analysis would also be helpful.

While these experiments in analysis were under way, Ugo Buy suggested a somewhat more global approach to generating inequalities and constructed a prototype tool implementing part of his approach. His idea is to regard a task expression as a graph, with the event symbols as the terminal nodes, and to generate a list of the paths leading to each symbol. Introducing auxiliary variables for the number of times each branch is traversed and generating inequalities involving these variables from the constraints and the semantics of regular expressions, one gets inequalities involving the number of occurrences of each symbol in a constrained prefix. Combined with some of the heuristics discovered in the course of the experiments described here, this approach seems very promising.

References

[1] G. S. Avrunin, L. K. Dillon, J. C. Wileden, and W. E. Riddle. Constrained expressions: adding analysis capabilities to design methods for concurrent software systems. *IEEE Transactions on Software Engineering*, SE-12(2):278–292, 1986.

[2] G. S. Avrunin and J. C. Wileden. Describing and analyzing distributed software system designs. *ACM Transactions on Programming Languages and Systems*, 7(3):380–403, July 1985.

[3] L. K. Dillon. *A Constrained Expression Formulation of CEDL*. Technical Report TRCS86-22, Department of Computer Science, University of California, Santa Barbara, November 1986. Revised July 1987.

[4] L. K. Dillon. *Overview of the Constrained Expression Design Language*. Technical Report TRCS86-21, Department of Computer Science, University of California, Santa Barbara, October 1986.

[5] L. K. Dillon. *Simplification and Reduction of CEDL Constrained Expressions*. Technical Report, Department of Computer Science, University of California, Santa Barbara, October 1986.

[6] L. K. Dillon, G. S. Avrunin, and J. C. Wileden. Constrained Expressions: Toward Broad Applicability of Analysis Methods for Distributed Software Systems. To appear in *ACM Transactions on Programming Languages and Systems*.

[7] D. Helmbold and D. Luckham. Debugging Ada tasking programs. *IEEE Software*, 2(2):47–57, March 1985.

[8] A. H. Land and S. Powell. *Fortran Codes for Mathematical Programming: Linear, Quadratic and Discrete*. John Wiley & Sons, Ltd., London, 1973.