ISBN 5-94052-127-4 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ. ПЕРЕСЛАВЛЬ-ЗАЛЕССКИЙ, 2006

УДК 681.32

А. П. Лисица, А. П. Немытых

Работа над ошибками

Аннотация. Ошибки в публикациях являются естественным продуктом человеческой жизнедеятельности. Некоторые из ошибок приводят к положительным последствиям. Мы рассматриваем две такие ошибки, обнаруженные автоматически специализатором функциональных программ SCP4. Мы начнем с простого примера и закончим удачной верификацией некоторого cache coherence протокола; то есть обнаружением ошибки в описании Xerox PARC Dragon протокола, данного в [4] и автоматическим доказательством корректности другого описания этого же протокола. Библ. 13 наим.

Ключевые слова и фразы: верификация, тестирование, специализация программ, суперкомпиляция. Cache coherence протоколы.

1. Введение

Автоматизация верификации вычислительных систем является актуальной задачей современной индустрии программирования. Следствием ошибок в описаниях вычислительных систем (программ, цифровых устройств или протоколов) в лучшем случае могут быть значительные потраченные впустую материальные средства, в худшем — человеческие жертвы.

Существует обширная литература посвященная данной проблеме. Нами проведен ряд удачных экспериментов по верификации параметрических cache coherence протоколов.

Мы предлагаем рассматривать верификацию как параметризованное тестирование. Пусть дана вычислительная система S. Пусть такты времени её эволюции помечены именами действий, которые

Работа выполнена при частичной поддержке Программы фундаментальных исследований Президиума РАН «Разработка фундаментальных основ создания научной распределенной информационно-вычислительной среды на основе технологий GRID».

изменяют систему S. Реализуем интерпретатор вычислительной системы S, который по данному времени `time` и начальной конфигурации $\texttt{InitConf}_0$ строит конфигурацию системы S, которой она будет обладать через время `time`

$$\texttt{Int(time, InitConf}_0\texttt{)},$$

тогда тестирование системы S в момент времени $\texttt{t}_0$ по данному постусловию φ есть верификация системы S в в момент времени $\texttt{t}_0$ по этому постусловию φ. То есть

$$\varphi \circ \texttt{Int(t}_0\texttt{, InitConf}_0\texttt{)} = \texttt{True}$$

Теперь рассмотрим оптимизацию предиката

$$\texttt{P}_\varphi \circ \texttt{Int(time, InitConf}_0\texttt{)},$$

где $\texttt{P}_\varphi$ программа реализующая проверку постусловия φ, а время `time` неизвестно (является параметром задачи). Если у нас есть оптимизатор, который сможет доказать, что

$$\forall\texttt{time}.(\texttt{P}_\varphi \circ \texttt{Int(time, InitConf}_0\texttt{)} = \texttt{True}),$$

то он проверифицирует по постусловию φ систему S. Так как в S такты времени помечены происходящими в S действиями, а утверждение постусловия верно $\forall$ `time` и, следовательно, для любой последовательности действий.

В качестве оптимизатора мы используем суперкомпилятор SCP4 (см. [8–10]), объектным языком которого является Рефал-5 (см. [12, 13]). Основная часть статьи посвящена удачному эксперименту верификации the Xerox PARC Dragon cache coherence протокола. Мы обнаружили ошибку в спецификации этого протокола данной в [4] и построили (в полуавтоматическом режиме) тест указывающий на эту ошибку. Другое описание Dragon протокола [5] оказалось корректным, что и доказал оптимизатор SCP4 методом математической индукции по времени эволюции протокола. Представляет интерес не только факт доказательства корректности, но и само доказательство. Мы подробно разбираем доказательство данное SCP4. Отметим, что SCP4 в процессе доказательства данного свойства корректности *обобщает* как гипотезы — промежуточные утверждения, так и само доказываемое утверждение корректности.

2. Малые чирки

На 19 странице сборника старинных занимательных задач [2] под номером 31 мы находим: «Хозяин послал работника на базар купить 20 птиц: гусей, уток и малых чирков. Он дал работнику 16 алтын[1]. Гусей велел купить по 3 копейки за штуку, уток по копейке , а малых чирков по два на копейку. Сколько гусей, сколько уток и сколько чирков купил работник?» На страницах 74 и 75 [2] дано решение поставленной задачи, которое завершается следующим предложением: «Таким образом, работник купил 15 гусей, 1 утку и 4 чирка.» Примем этот ответ к сведению.

Перепишем условие задачи в более формальных терминах, стараясь не нарушить структуру формулировки. Все вычисления будем производить посредством счетных палочек. Равно по определению, — запишем так:

```
Altyn { = I I I; }
Birds { = I I I I I    I I I I I   I I I I I   I I I I I; }
Money{ = <Altyn> <Altyn> <Altyn> <Altyn> <Altyn> <Altyn> <Altyn> <Altyn>
         <Altyn> <Altyn> <Altyn> <Altyn> <Altyn> <Altyn> <Altyn> <Altyn>;}
```

где угловыми скобками мы обозначили конструктор применения одноточечной функции `Altyn` (т.е. функции, определенной только в одной точке).

Понятие половины определим индуктивно, где переменную типа натуральное число мы обозначим `e.n`, подчеркивая унарность нашего представления натуральных чисел:

```
Half { I I e.n = I <Half e.n>;
               = ;
}
*Аналогично имеем:
Three { I  e.n = I I I <Three e.n>;
               = ; }
```

Вместо предикатов мы рассмотрим частичные константные функции со значением `True`: их область определения есть область истинности соответствующего предиката. В «предикате» равенства аргументы заключим в скобки:

[1]Алтын — три копейки.


```
Eq {
 (I e.n) (I e.m) = <Eq (e.n) (e.m)>;
 ()          () = True; }
```

Здесь, через `True` мы обозначили палочку, помеченную соответствующим словом. Нужна также логическая связка

```
And { True True = True; }
```

Теперь мы готовы описать вопрос, поставленный в задаче, в виде следующей композиции частичных функций. Найти множество определения частичной функции–константы:

```
$ENTRY Shopping {
 (Gees e.gees) (Ducks e.ducks) (Teals e.teals) =
               <And <Eq (e.gees e.ducks e.teals) (<Birds>)>
                    <Eq (<Three e.gees> e.ducks <Half e.teals>)(<Money>)>
               >; }
```

где для каждого вида птицы мы выделили отдельную корзину.

Мы описали задачу в терминах функционального языка программирования РЕФАЛ [12] и непосредственно формулировку выделили ключевым словом `$ENTRY`. Если мы оптимизируем построенную программу посредством автоматического оптимизатора–специализатора РЕФАЛ программ SCP4 [8–10], то получим:

```
$ENTRY Shopping {
(Gees I I I I I I I I I I I I I I)(Ducks I)(Teals I I I I)    = True;
(Gees I I I I I I I I I I I I I I)  (Ducks I I I I I I)(Teals ) = True;
}
```

То есть, область определения частичной функции `Shopping` не одноточечная, как это утверждается в [2], а двухточечная.

3. The Xerox PARC Dragon cache coherence протокол

В мультипроцессорных системах с разделяемой основной памятью используются также небольшие элементы памяти (caches), доступ к данным в которых производится существенно быстрее чем к данным в основной памяти. Процессоры используют эти элементы для работы с копиями блоков основной памяти. При этом обычное требование состоит в том, что кратные копии одного и того же блока

должны совпадать в любой момент времени работы системы. Протоколы когерентности кэш–памяти призваны гарантировать такое совпадение. Описание некоторых из таких протоколов с условиями глобальной корректности может быть сформулировано [4] ввиде игр — недетерминированного перекладывания камешков из одних корзин в другие. А корректность таких протоколов выражается системами линейных неравенств на количества камней в корзинах.

Модель с корзинами и камнями представляет собой абстракцию автоматной модели протоколов. Здесь камешки следует понимать как экземпляры автоматов, а их присутствие в той или иной корзине M — как представление того факта, что автомат находится в состоянии M. Детали см. в [4].

Пусть даны пять корзин, которые мы будем обозначать скобками:

(invalid x_1)(shared_clean x_2)(shared_dirty x_3)(dirty x_4)(exclusive x_5)

где x_i — число камней в соответствующей корзине (иногда мы будем обозначать их именами корзин). Ж. Дельзанно [4] описывает игру Dragon нижеследующим образом:

```
(rh) dirty + shared_clean + exclusive + shared_dirty ≥ 1 → ·
(rm1) invalid ≥ 1,
      dirty = shared_clean = shared_dirty = exclusive = 0 →
             invalid' = invalid - 1, exclusive' = 1 + exclusive.
(rm2) invalid ≥ 1,
      shared_clean + shared_dirty + exclusive + dirty ≥ 1 →
             invalid' = invalid - 1, exclusive' = 0,
             shared_clean' = 1+exclusive + shared_clean,
             shared_dirty' = dirty + shared_dirty, dirty = 0.
(wm1) invalid ≥ 1,
      dirty = shared_clean = shared_dirty = exclusive = 0 →
             invalid' = invalid - 1, dirty' = 1 + dirty.
(wm2) invalid ≥ 1,
      shared_clean + shared_dirty + exclusive + dirty ≥ 1 →
         invalid' = invalid - 1, exclusive' = 0, shared_dirty' = 1
         shared_clean' = shared_dirty + exclusive + shared_clean.
(wh1) dirty ≥ 1 → ·
(wh2) exclusive ≥ 1 →
             exclusive' = exclusive - 1, dirty' = 1 + dirty.
(wh3) shared_dirty = 1, shared_clean = 0 →
             shared_dirty' = 0, dirty' = 1 + dirty.
(wh4) shared_dirty = 0, shared_clean = 1 →
             shared_clean' = 0, dirty' = 1 + dirty.
(wh5) shared_dirty + shared_clean ≥ 2 → shared_dirty' = 1,
             shared_clean' = shared_clean + shared_dirty - 1.
```


Начальная конфигурация игры:

(invalid 1+x_1)(shared_clean 0)(shared_dirty 0)(dirty 0)(exclusive 0)

Игра состоит в случайном применении этих правил к текущей конфигурации; если в текущей конфигурации левая часть правила выполнена, то производятся преобразования описанные в правой части этого правила.

Корректность приведенного выше описания протокола Dragon, по мнению Ж. Дельзанно [4], должна выражаться в недостижимости конфигураций вида:

- (C_1) dirty ≥ 2;
- (C_2) exclusive ≥ 2;
- (C_3) exclusive+shared_clean+shared_dirty ≥ 1, dirty ≥ 1;
- (C_4) exclusive ≥ 1, shared_clean + shared_dirty ≥ 1.

В вышеприведенном описании протокола Dragon имеется ошибка. При конкретной начальной конфигурации

(invalid 3)(shared_clean 0)(shared_dirty 0)(dirty 0)(exclusive 0).

последовательность шагов wm_1, wm_2, wh_3 игры Dragon приводит к конфигурации вида C_1, где dirty = 2. Тест был построен после неудачной попытки автоматической верификации описания протокола, данного выше, специализатором SCP4. Более того, тест был построен в результате анализа РЕФАЛ программы построенной SCP4.

Прежде всего, опишем сам метод верификации. Он основан на понимании верификации как параметризованного тестирования; и в нашем случае тестирование проведено удачно — предъявлен тест, указывающий на ошибку.

4. Верификация как параметризованное тестирование

Пусть $\mathtt{P}_\varphi$ — программа, реализующая предикат φ, а $\mathtt{P}_f$ — программа, для любого $\mathtt{d}_0 \in D$ завершающая исполнение вызова $\mathtt{P}_f(\mathtt{d}_0)$ в конечное время, и о которой предполагается что она реализует f. Тогда тестированием программы $\mathtt{P}_f$ по постусловию φ назовем программу

$$T : D \mapsto \{\mathtt{True},\ \mathtt{False}\}$$

реализующую композицию $\mathtt{P}_\varphi \circ \mathtt{P}_f$. Для любого конкретного $\mathtt{d}_0 \in D$ результат вычисления $T(\mathtt{d}_0) = \mathtt{True}$ подтверждает корректность $\mathtt{P}_f$ на выбранном тесте $\mathtt{d}_0$, а $T(\mathtt{d}_0) = \mathtt{False}$ дает тест $\mathtt{d}_0$, на котором корректность нарушается.

Перебор всего множества D с положительным результатом тестирования гарантирует корректность $\mathtt{P}_f$ по постусловию φ, то есть верифицирует $\mathtt{P}_f$ по постусловию φ.

Если D бесконечно, то прямой перебор невозможен. Предположим, что у нас есть оптимизирующая программа Ω, результат вычисления которой $\Omega(\underline{\mathtt{P}_\varphi \circ \mathtt{P}_f}, d)$ есть программа π (здесь подчеркивание означает кодировку, а d — формальный параметр преобразуемой программы), обладающая простым *с*интаксическим свойством, позволяющим утверждать, что $Im(\pi) = \{\mathtt{True}\}$. Пусть результат оптимизации, по определению Ω, всегда реализует некоторое расширение частичной функции, которая реализована преобразуемой программой (в нашей ситуации программой $\mathtt{P}_\varphi \circ \mathtt{P}_f$). В таком случае мы получаем автоматическую верификацию $\mathtt{P}_f$ по постусловию φ.

Описанию экспериментов по описанной схеме мы и посвятим остаток данной статьи. В качестве инструмента Ω мы используем суперкомпилятор SCP4 [8, 10].

5. Интерпретатор игры Dragon

Игра Dragon является недетерминированной динамической системой с дискретным временем. Такты времени случайным образом помечены именами действий (см. описание данное выше). Мы смоделируем эту динамическую систему добавлением к конфигурации игры дополнительной корзины (ниже мы иногда будем сокращать имена корзин)

`(time t)(inv` x_1`)(shc` x_2`)(shd` x_3`)(dirty` x_4`)(exc` x_5`)`

Будем выбирать из корзины time шары помеченные именами действий и выполнять обнаруженное действие. Шагом игры назовем либо выбор действия из `time` (если `time` не пуста, или установление факта, что `time` пуста), либо исполнение выбранного действия, либо проверку предиката $\mathtt{Test}(x_1, x_2, x_3, x_4, x_5) = \neg(C_1 \vee C_2 \vee C_3 \vee C_4)$ на текущей конфигурации — в случае, когда время закончилось. Пусть k — число шагов в конкретной игре, тогда $k = 2t + 1$.

Данная динамическая система адекватно описывается на языке Рефал. Пример программы на Рефале мы уже показали в 2. Дадим некоторые пояснения к Рефалу в рамках кодировки нашей игры. Подробности см. в [12].

5.1. Кодировка (вариант первый). Семантика Рефала основана на отождествлении по образцу. Грубо говоря, программа есть система переписывания термов. Отождествление перебирает левые части правил переписывания сверху вниз до первой удачи. Конструктор применения (вызова) функции обозначается угловыми скобками `<F ...>`, где имя вызываемой функции следует непосредственно после открывающей скобки. Все функции в Рефале формально имеют только один аргумент. Данные Рефала определяются грамматикой: `d ::=` $(\mathtt{d}_1)$ `|` $\mathtt{d}_1\ \mathtt{d}_2$ `| SYMBOL |` *empty*, где *empty* — пустая строка; пробел представляет ассоциативную операцию приписывания (*empty* — ее единица); `SYMBOL`, в нашем случае, — идентификатор. Нам понадобятся только два типа переменных: переменные типа `'e'` — `e.name`, область допустимых значений которых — все множество Рефал–данных; переменные типа `'s'` — s.name, О.Д.З. которых — множество символов. Звездочкой в начале строки обозначаются комментарии.

Входная точка программы помечается ключевым словом `$ENTRY`. Чтобы имитировать второй аргумент входной точки `Go`, мы заключаем его в скобки. Функция `Test` проверяет условие корректности конфигурации игры, когда время игры `e.time` закончилось.

```
*$MST_FROM_ENTRY;
*$STRATEGY Applicative;
*$LENGTH 0;
$ENTRY Go {
 e.time (e.x1) = <Loop (time e.time)(invalid I e.x1)
                       (shared_clean)(shared_dirty)(dirty)(exclusive)>;
}
Loop {
(time )(invalid e.x1)(shared_clean e.x2)
       (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5)
     = <Test (invalid e.x1)(shared_clean e.x2)
             (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5)>;
(time s.tm e.t)(invalid e.x1)
       (shared_clean e.x2)(shared_dirty e.x3)(dirty e.x4)(exclusive e.x5)
    =  <Loop (time e.t) <RandomAction s.tm
          (invalid e.x1)(shared_clean e.x2)
          (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5)
       >>;
}

RandomAction {
* rh Trivial
rm1 (invalid I e.x1)(shared_clean )(shared_dirty )(dirty )(exclusive )
    = (invalid e.x1)(shared_clean )(shared_dirty )(dirty )(exclusive I);
rm2A (invalid I e.x1)(shared_clean I e.x2)(shared_dirty e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean I e.x5 I e.x2)
                        (shared_dirty e.x4 e.x3)(dirty )(exclusive );
rm2B (invalid I e.x1)(shared_clean e.x2)(shared_dirty I e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean I e.x5 e.x2)
                        (shared_dirty e.x4 I e.x3)(dirty )(exclusive );
rm2C (invalid I e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty I e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean I e.x5 e.x2)
                        (shared_dirty I e.x4 e.x3)(dirty )(exclusive );
rm2D (invalid I e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty e.x4)
     (exclusive I e.x5) = (invalid e.x1)(shared_clean I I e.x5 e.x2)
                          (shared_dirty e.x4 e.x3)(dirty )(exclusive );
wm1 (invalid I e.x1)(shared_clean )(shared_dirty )(dirty )(exclusive )
    = (invalid e.x1)(shared_clean )(shared_dirty )(dirty I)(exclusive );
```



```
* shared_clean + shared_dirty + dirty + exclusive >= 1
wm2A (invalid I e.x1)(shared_clean I e.x2)(shared_dirty e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean e.x3 e.x5 I e.x2)
                        (shared_dirty I)(dirty e.x4)(exclusive );
wm2B (invalid I e.x1)(shared_clean e.x2)(shared_dirty I e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean I e.x3 e.x5 e.x2)
                        (shared_dirty I)(dirty e.x4)(exclusive );
wm2C (invalid I e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty I e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean e.x3 e.x5 e.x2)
                        (shared_dirty I)(dirty I e.x4)(exclusive );
wm2D (invalid I e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty e.x4)
     (exclusive I e.x5) = (invalid e.x1)(shared_clean e.x3 I e.x5 e.x2)
                          (shared_dirty I)(dirty e.x4)(exclusive );
*wh1 Trivial
wh2 (invalid e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty e.x4)
    (exclusive I e.x5)= (invalid e.x1)(shared_clean e.x2)
                        (shared_dirty e.x3)(dirty I e.x4)(exclusive e.x5);
wh3 (invalid e.x1)(shared_clean )(shared_dirty I)(dirty e.x4)
    (exclusive e.x5) = (invalid e.x1)(shared_clean )
                       (shared_dirty )(dirty I e.x4)(exclusive e.x5);
wh4 (invalid e.x1)(shared_clean I)(shared_dirty )(dirty e.x4)
    (exclusive e.x5) = (invalid e.x1)(shared_clean )
                       (shared_dirty )(dirty I e.x4)(exclusive e.x5);
wh5A (invalid e.x1)(shared_clean I I e.x2)(shared_dirty e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean e.x3 I e.x2)
                        (shared_dirty I)(dirty e.x4)(exclusive e.x5);
wh5B (invalid e.x1)(shared_clean I e.x2)(shared_dirty I e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean e.x3 I e.x2)
                        (shared_dirty I)(dirty e.x4)(exclusive e.x5);
wh5C (invalid e.x1)(shared_clean e.x2)(shared_dirty I I e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid e.x1)(shared_clean I e.x3 e.x2)
                        (shared_dirty I)(dirty e.x4)(exclusive e.x5);
}

Test {
* C1.
(invalid e.x1)(shared_clean e.x2)
              (shared_dirty e.x3)(dirty I I e.x4)(exclusive e.x5) = False;
* C2.
(invalid e.x1)(shared_clean e.x2)
              (shared_dirty e.x3)(dirty e.x4)(exclusive I I e.x5) = False;
* C3.
(invalid e.x1)(shared_clean)(shared_dirty)(dirty I e.x4)(exclusive)= True;
```

```
(invalid e.x1)(shared_clean e.x2)
             (shared_dirty e.x3)(dirty I e.x4)(exclusive e.x5) = False;
* C4.
(invalid e.x1)(shared_clean )
             (shared_dirty )(dirty e.x4)(exclusive I e.x5) = True;
(invalid e.x1)(shared_clean e.x2)
             (shared_dirty e.x3)(dirty e.x4)(exclusive I e.x5) = False;

(invalid e.x1)(shared_clean e.x2)
             (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5) = True; }
```

5.2. Поиск ошибки. Оптимизатор SCP4 упростил описанную выше кодировку протокола до остаточной программы, в которой мы видим, что ее формальный синтаксис не позволяет сделать вывод о сужении образа нашего предиката до одной точки (в правых частях предложений присутствуют `True` и `False`). Сожалеем мы, на первых порах, о слабости оптимизатора SCP4, «который не смог доказать корректности протокола Dragon». Не все константы `False` в остаточной программе лежат на достижимых ветках. Проиндексируем константу `False` номером I конфигурации, которой она соответствует (см. исходный текст в разделе 5.1), — `False`I. В этом варианте остаточная программа содержит только `False1` и `False3`.

Рассмотрим первый случай. Конфигурация ему соответствующая обладает свойством `dirty` $\geq$ 2. Попытаемся автоматизировать анализ причин существования `False` в остаточной программе. Функцию `Test` можно упростить, оставив в ней только первое и последнее предложения. Результат оптимизации упрощенной программы также содержит `False`. Рассмотрим конкретное конечное время развития игры. Начнем с двух тактов `e.time = s.t1 s.t2`. Результат суперкомпиляции в этом случае не содержит `False`. Пусть `e.time = s.t1 s.t2 s.t3`. Теперь остаточная программа схематично выглядит так:

```
* InputFormat: <Go s.t1 s.t2 s.t3 (e.x1)>
$ENTRY Go {
rm1 rm2D rm2A (I I e.x1) = True ;
.........................
s.t1 s.t2 wm2C (I I e.x1) = True ;
s.t1 s.t2 s.t3 (I e.x1) = False ; }
```

Мы видим, что либо существует трехтактовый тест указывающий на

ошибку в описании протокола, либо наш оптимизатор не справляется с ситуацией в рассматриваемом простейшем случае.

Из `dirty` $\geq$ 2 и формального синтаксиса исходной программы следует, что в тесте на некорректность в процессе игры необходимо должен быть вытянут шар с одной из следующих меток: `wh2, wh3, wh4`. Эксперименты с SCP4 показывают, что при `s.t1 = wh2` и `s.t3 = wh2` игра Dragon не реализуема, а при `s.t2 = wh2`, протокол корректен. Аналогично, рассматривая случай `wh3`, при `s.t3 = wh3` имеем остаточную программу: `$ENTRY Go { wm1 wm2C (I e.x1) = False; }`. Таким образом, построена последовательность тактов `wm1 wm2C wh3`, которая может оказаться тестом на некорректность протокола. Исполнение исходной программы при стартовой конфигурации

```
<Loop (time wm1 wm2C wh3)(invalid I I I)(shared_clean)
                         (shared_dirty)(dirty)(exclusive)>
```

приводит к результату `False`. Следовательно, протокол, как он описан выше некорректен — уже исполнение трех тактов может вести к нарушению условий корректности. Вспоминая, что `wm2C` есть подслучай `wm2`, убеждаемся, что либо описание протокола Dragon данное в разделе 3 содержит ошибку, либо утверждение автора о корректности неверно.

6. Корректная версия the Xerox PARC Dragon протокола

Двигаясь по ссылкам данным в [4], мы нашли другое описание протокола Dragon в [5]. Ниже приведены те случаи, где оно отличается от описания данного в разделе 3.

```
(wm2) invalid ≥ 1,
      shared_clean + shared_dirty + exclusive + dirty ≥ 1 →
         invalid' = invalid - 1, exclusive' = 0, dirty' = 0,
         shared_clean' = shared_dirty + dirty +
                         + exclusive + shared_clean.
(wh1) dirty ≥ 1 → invalid' = 1 + invalid, dirty' = dirty - 1.
(wh2) shared_clean ≥ 1 → invalid' = 1 + invalid,
              shared_clean' = shared_clean - 1.
(wh3) shared_dirty ≥ 1 → invalid' = 1 + invalid,
              shared_dirty' = shared_dirty - 1.
(wh4) exclusive ≥ 1 → invalid' = 1 + invalid,
```

```
                exclusive' = exclusive - 1.
(wh5) — этого случая теперь нет.
```

Это описание протокола оказалось корректным, — что и установил наш оптимизатор. Соответствующие изменения в кодировке выглядят следующим образом:

```
RandomAction {
  ..................
* wm2 corrected version.
* shared_clean + shared_dirty + dirty + exclusive >= 1
 wm2A (invalid I e.x1)(shared_clean I e.x2)(shared_dirty e.x3)(dirty e.x4)
      (exclusive e.x5)= (invalid e.x1)(shared_clean e.x3 e.x4 e.x5 I e.x2)
                        (shared_dirty I)(dirty )(exclusive );
 wm2B (invalid I e.x1)(shared_clean e.x2)(shared_dirty I e.x3)(dirty e.x4)
      (exclusive e.x5)= (invalid e.x1)(shared_clean I e.x3 e.x4 e.x5 e.x2)
                        (shared_dirty I)(dirty )(exclusive );
 wm2C (invalid I e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty I e.x4)
      (exclusive e.x5)= (invalid e.x1)(shared_clean e.x3 I e.x4 e.x5 e.x2)
                        (shared_dirty I)(dirty )(exclusive );
 wm2D (invalid I e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty e.x4)
   (exclusive I e.x5) = (invalid e.x1)(shared_clean e.x3 e.x4 I e.x5 e.x2)
                        (shared_dirty I)(dirty )(exclusive );
*wh corrected version.
 wh1 (invalid e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty I e.x4)
     (exclusive e.x5) = (invalid I e.x1)(shared_clean e.x2)
                        (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5);
 wh2 (invalid e.x1)(shared_clean I e.x2)(shared_dirty e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid I e.x1)(shared_clean e.x2)
                        (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5);
 wh3 (invalid e.x1)(shared_clean e.x2)(shared_dirty I e.x3)(dirty e.x4)
     (exclusive e.x5) = (invalid I e.x1)(shared_clean e.x2)
                        (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5);
 wh4 (invalid e.x1)(shared_clean e.x2)(shared_dirty e.x3)(dirty e.x4)
     (exclusive I e.x5) = (invalid I e.x1)(shared_clean e.x2)
                          (shared_dirty e.x3)(dirty e.x4)(exclusive e.x5);
*wh5 - this case is removed.
}
```

6.1. Схема доказательства корректности Xerox Dragon протокола. Кроме игры с данной стартовой конфигурации:

$$[I_1]\texttt{: (inv } 1+x_1\texttt{) (shc 0) (shd 0) (dirty 0) (exc 0).}$$

рассмотрим также игру с начальной конфигурацией:

$$[I_2]\texttt{: (inv } x_1\texttt{) (shc } 2+x_2\texttt{) (shd 0) (dirty 0) (exc 0).}$$

Теорема 1. *Игры Dragon начинающиеся с конфигураций $[I_1]$ и $[I_2]$ не достигают конфигураций вида (C_i), для всех $1 \le i \le 4$.*

Докажем эту теорему. Для нас будет важен не факт доказательства, а его способ и структура. Именно *это* доказательство (с точностью до несущественных деталей) и приводит SCP4, если сами игры закодированы в виде Рефал программ, данных нами выше [12, 13].

Перепишем утверждение теоремы на более формальном языке:

$$(\forall t\ \forall x_1\ P(2t+1, 1+x_1, 0))\ \&\ (\forall t\ \forall x_1 \forall x_2\ P(2t+1, x_1, 2+x_2))$$

где $t, x_1, x_3 \in \mathbb{N}, P$ — утверждение теоремы, относящееся к игре с конкретной начальной конфигурацией с временем игры t, а последний аргумент указывает на тип начальной конфигурации. Пусть

$$P_1(t, x_1, x_2) := P(2t+1, 1+x_1, 0),\quad P_2(t, x_1, x_2) := P(2t+1, x_1, 2+x_2).$$

Мы будем вести индукцию по t. Наибольший интерес представляет шаг индукции ($l = 1, 2$):

$$\forall t((\forall n > 0\ (\forall x_1 \forall x_2\ P_l(t-n, x_1, x_2))) \Rightarrow (\forall x_1 \forall x_2\ P_l(t, x_1, x_2))).$$

Наш план такой: стартуя с $\forall x_1 \forall x_2\ P_l(m, x_1, x_2) \quad (m > 0)$ будем следить за развитием обеих динамических систем (соответствующих конфигурациям $[I_1]$ и $[I_2]$), спускаясь по времени (и более детально — по шагам). И будем подыскивать множество пар (t_i, n_i) и пар (t_j, k_j) таких, что $m = t_i + n_i = t_j + k_j,\ n_i > 0,\ k_j > 0$ и

$$(\&_i(\forall x_1 \forall x_2 P_1(t_i, x_1, x_2)) \& (\&_j(\forall x_1 \forall x_2 P_2(t_j, x_1, x_2)) \Rightarrow (\forall x_1 \forall x_2 P_l(m, x_1, x_2)))$$

Здесь мы ослабляем посылку шага схемы индукции, и, следовательно, доказываем более сильное утверждение.

Одновременно будут обнаруживаться случаи, когда время исчерпано. Они и показывают базис индукции, который доказывается по ходу дела. Перебор дерева возможных случаев будет вестись вглубь (depth first). Арифметические действия будем проводить в унарной системе счисления, представляя положительное число m, согласно с нашей игрой, камнями (шарами): 1 `::= I,` $n+1$ `::= I` n. И 0 — пустой строкой (иногда — условно самим нулем). Сложение в такой

системе есть приписывание строк. Свойство, описывающее недостижимые конфигурации не зависит от значения x_1: `Test(`x_2`,`x_3`,`x_4`,`x_5`)`.

Заметим, что указанная выше индукция есть частный случай Нетеровской схемы индукции

$$\frac{\forall x.(\forall y.\ (y \prec x) \to P(y)) \to P(x)}{\forall x.P(x)}$$

где $\prec$ есть какое-либо отношение вполне-упорядочения [3]. В нашем случае в роли $\prec$ выступает отношение (обратной) достижимости между конфигурациями игры (протокола). Т.е. $y \prec x$ имеет место если и только если y достижимо из x за ненулевое число шагов. Поскольку первая компонента конфигурации — время оставшееся до конца исполнения, то очевидно, что не существует бесконечных цепей вида

$$a_1 \succ a_2 \succ a_3 \succ \ldots \succ a_n \succ \ldots$$

что требуется для вполне–упорядочевания наряду с нерефлексивностью и антисимметричностью. Поиск доказательства корректности протокола Dragon, использующий такую схему далеко нетривиален и включает шаги *обобщения* промежуточных утверждений, да и самого доказываемого утверждения.

В рассуждениях мы будем придерживаться более привычной терминологии описания, а не закодированным ее вариантом.

6.2. Доказательство корректности протокола Dragon.

6.2.1. *Развитие* $\forall x_1\ P(2m+1, 1+x_1, 0)$. Конфигурация

`[1]: (time` m`) (inv` $1+x_1$`) (shc )(shd )(dirty )(exc ).`

Если m `= 0`, то `Test(0,0,0,0) = True`, иначе m `= I` m_1, вынимаем шар t_m из корзины `time`, пометим этот шаг как `[2]` и перебираем все возможные его метки. Если t_m `= rm1`, тогда следующая конфигурация имеет вид

`[3]: (time` m_1`) (inv` x_1`)(shc )(shd )(dirty )(exc I).`

Будем следить за развитием конфигурации `[3]`, отложив разбор оставшихся случаев $t_m \in \{$`rm2, wm1, wm2, wh`$_i$ $(1 \le i \le 4)\}$ для шага `[2]` на более поздний этап. Т.е. разбираем дерево ветвлений вглубь.

Если $m_1 = 0$, то имеем еще один базисный случай индукции $(m = 1)$ и `Test(0,0,0,I) = True`, иначе m_1 `= I` m_2, вынимаем шар

t_{m_1} из корзины `time`, пометим этот шаг как `[4]` и перебираем значения t_{m_1}. Шар t_{m_1} `= rm1` требует условия `exclusive = 0`, противоречащего текущей конфигурации, и потому не приводит к развитию игры.

Если t_{m_1} `= rm2`, тогда необходимо x_1 `= I` x_{11} и следующая конфигурация имеет вид

`[5]: (time` m_2`) (inv` x_{11}`)(shc I I)(shd )(dirty )(exc ).`

Ей соответствует предикат $\forall x_{11}\ P(2m_2+1, x_{11}, 2)$, ему и посвятим следующий раздел, предварительно отложив разбор случаев `wm1, wm2, wh`$_i$ $(1 \le i \le 4)$ для шага `[4]`.

6.2.2. *Развитие* $\forall x_1 \forall x_2\ P(2m_2+1, x_1, 2+x_2)$. Мы рассмотрим более общее утверждение, данное в заголовке, и будем его доказывать индукцией по m_2. Конфигурация игры:

`[5g]: (time` m_2`) (inv` x_1`)(shc I I` x_2`)(shd )(dirty )(exc ).`

Если $m_2 = 0$, то имеем базисный случай индукции $m = 2$ и $\forall x_2$ `Test(2+`x_2`,0,0,0) = True`, иначе m_2 `= I` m_3, вынимаем шар t_{m_2} из корзины `time`, пометим этот шаг как `[6]` и перебираем значения t_{m_2}. Шар t_{m_2} `= rm1` не приводит к развитию игры.

Если t_{m_2} `= rm2`, тогда необходимо x_1 `= I` x_{12}, и следующая конфигурация: `(time` m_3`)(inv` x_{12}`)(shc I I I` x_2`)(shd )(dirty )(exc ).`

Ей соответствует предикат $\forall x_{12} \forall x_2\ P(2m_3+1, x_{12}, 3+x_2)$, где $m_3 < m_2$, и он истинен по предположению индукции данного раздела. Случай t_{m_2} `= wm1` не реализуется.

Если t_{m_2} `= wm2`, тогда необходимо x_1 `= I` x_{13} и следующая конфигурация имеет вид

`[7]: (time` m_3`) (inv` x_{13}`)(shc I I` x_2`)(shd I)(dirty )(exc ).`

Будем следить за развитием конфигурации `[7]`, отложив разбор случаев `wh`$_i$ $(1 \le i \le 4)$ для шага `[6]`.

Если $m_3 = 0$, то имеем базисный случай индукции $(m = 3)$ и $\forall x_2$ `Test(2+`x_2`,1,0,0) = True`, иначе m_3 `= I` m_4, вынимаем шар t_{m_3} из корзины `time`, пометим этот шаг как `[8]` и перебираем значения t_{m_3}. Шар t_{m_3} `= rm1` не приводит к развитию игры.

Если t_{m_3} `= rm2`, тогда необходимо x_{13} `= I` x_{14} и следующая конфигурация имеет вид

`[*]: (time` m_4`) (inv` x_{14}`)(shc I I I` x_2`)(shd I)(dirty )(exc ).`

Похожую конфигурацию мы уже рассматривали — это [7]. Конфигурации [*], [7] описывают состояние игры в соответствующие моменты времени m_4 и $m_3 = m_4 + 1 > m_4$.

Лемма 1. *Игра Dragon начинающаяся с конфигурации [7] никогда не достигает конфигураций вида (C_i), для всех $1 \le i \le 4$. Пусть t — время, x_1 — содержимое корзины* **invalid**, *$2 + x_2$ — содержимое корзины* **shared_clean**, *тогда обозначим утверждение леммы так $\forall t\ \forall x_1\ \forall x_2\ Q_1(2t + 1, x_1, 2 + x_2)$.*

Если наше доказательство рассматривать с точки появления конфигурации [7] ($t = m_3$), тогда оно является доказательством предиката $\forall \tau\ \forall x_{13}\ \forall x_2\ Q_1(2\tau+1, x_{13}, 2+x_2)$ индукцией по τ. Далее, конфигурации [*] соответствует предикат $\forall x_{14}\ \forall x_2\ Q_1(2m_4+1, x_{14}, 3+x_2)$, где $m_4 < m_3 < m$, и он истинен по предположению индукции леммы 1. Доказательство случая t_{m_3} = rm2 шага [8] закончено. Переходим к случаю t_{m_3} = wm1, — он тупиковый.

Рассмотрим t_{m_3} = wm2. В этом случае необходимо x_{13} = I x_{15} и следующая конфигурация имеет вид

(time m_4) (inv x_{15})(shc I I I x_2)(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_{15}\ \forall x_2\ Q_1(2m_4 + 1, x_{15}, 3 + x_2)$, где $m_4 < m_3 < m$, и он истинен по предположению индукции леммы 1. Шар t_{m_3} = wh1 не приводит к развитию игры.

Если t_{m_3} = wh2, тогда следующая конфигурация имеет вид

[9]: (time m_4) (inv I x_{13})(shc I x_2)(shd I)(dirty)(exc).

Далее следим за развитием [9], отложив разбор случаев wh3, wh4 для шага [8].

Если $m_4 = 0$, то имеем базисный случай индукции ($m = 4$) и $\forall x_2$ Test($1+x_2, 1, 0, 0$) = True, иначе m_4 = I m_5, вынимаем шар t_{m_4} из корзины time, пометим этот шаг как [10] и перебираем значения t_{m_4}. Шар t_{m_4} = rm1 не приводит к развитию игры.

Если t_{m_4} = rm2, тогда следующая конфигурация имеет вид

(time m_5) (inv x_{13})(shc I I x_2)(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_{13}\ \forall x_2\ Q_1(2m_5 + 1, x_{13}, 2 + x_2)$, где $m_5 < m_3 < m$, и он истинен по предположению индукции леммы 1. Шар t_{m_4} = wm1 не приводит к развитию игры.

Если t_{m_4} = wm2, тогда следующая конфигурация имеет вид

(time m_5) (inv x_{13})(shc I I x_2)(shd I)(dirty)(exc).

И, как и в случае t_{m_4} = rm2, он истинен по предположению индукции леммы 1. Шар t_{m_4} = wh1 не приводит к развитию игры.

Если t_{m_4} = wh2, тогда следующая конфигурация имеет вид

[11]: (time m_5) (inv I I x_{13})(shc x_2)(shd I)(dirty)(exc).

Будем следить за развитием [11], отложив разбор случаев wh3, wh4 для шага [10].

Если $m_5 = 0$, то имеем базисный случай индукции ($m = 5$) и $\forall x_2$ Test($x_2, 1, 0, 0$) = True, иначе m_5 = I m_6, вынимаем шар t_{m_5} из корзины time, пометим этот шаг как [12] и перебираем значения t_{m_5}. Шар t_{m_5} = rm1 не приводит к развитию игры.

Случай t_{m_5} = rm2 разобьем на подслучаи[2]:

(rm2A) shared_clean $\ge$ 1, (rm2B) shared_dirty$\ge$1,
(rm2C) dirty $\ge$ 1, (rm2D) exclusive $\ge$ 1 .

Это разбиение соответствует разбиению в кодировке (см. 5.1).

Если t_{m_5} = rm2A, тогда необходимо x_2 = I x_{21}, и следующая конфигурация:

(time m_6)(inv I x_{13})(shc I I x_{21})(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_{13}\ \forall x_{21}\ Q_1(2m_6 + 1, 1 + x_{13}, 2 + x_{21})$, где $m_6 < m_3 < m$, и он истинен по предположению индукции леммы 1.

Если t_{m_5} = rm2B, тогда следующая конфигурация имеет вид

[**]: (time m_6) (inv I x_{13})(shc I x_2)(shd I)(dirty)(exc).

Похожую конфигурацию мы уже рассматривали, напомним ее

[9]: (time m_4) (inv I x_{13})(shc I x_2)(shd I)(dirty)(exc).

[2] Ранее мы не разбивали вариант выбора шара rm2 на подслучаи. Причина заключается в том, что, при аналогичном разбиении, доказательство случаев развивающих игру полностью совпадало. Таким образом, мы пытаемся сделать изложение доказательства более обозримым; суперкомпилятор, рассматривает все случаи отдельно.

Эти конфигурации описывают состояние игры в соответствующие моменты времени m_6 и $m_4 = m_6 + 2 > m_6$.

Лемма 2. *Игра Dragon начинающаяся с конфигурации* **[9]** *никогда не достигает конфигураций вида* (C_i)*, для всех* $1 \le i \le 4$*. Пусть* t *— время,* $1 + x_1$ *— содержимое корзины* **invalid***,* $1 + x_2$ *— содержимое корзины* **shared_clean***, тогда обозначим утверждение леммы так* $\forall t\ \forall x_1\ \forall x_2\ Q_2(2t + 1, 1 + x_1, 1 + x_2)$.

Если наше доказательство рассматривать с точки появления конфигурации [9] ($t = m_4$), тогда оно является доказательством предиката $\forall\tau\ \forall x_{13}\ \forall x_2\ Q_2(2\tau+1, 1+x_{13}, 1+x_2)$ индукцией по τ. Конфигурации [**] соответствует предикат $\forall x_{13}\ \forall x_2\ Q_2(2m_6+1, 1+x_{13}, 1+x_2)$, где $m_5 < m_4 < m$, и он истинен по предположению индукции леммы 2. Шары $t_{m_5} \in \{$rm2C, rm2D, wm1$\}$ не приводят к развитию игры.

Случай t_{m_5} = wm2 разобьем на подслучаи[3]:

(wm2A) shared_clean $\ge$ 1, (wm2B) shared_dirty $\ge$ 1,
(wm2C) dirty $\ge$ 1, (wm2D) exclusive $\ge$ 1.

Это разбиение соответствует разбиению в кодировке (см. 6).

Если t_{m_5} = wm2A, тогда необходимо x_2 = I x_{22} и следующая конфигурация:

(time m_6)(inv I x_{13})(shc I I x_{22})(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_{13}\ \forall x_{22}\ Q_1(2m_6+1, 1+x_{13}, 2+x_{22})$, где $m_6 < m_3 < m$, и он истинен по предположению индукции леммы 1.

Если t_{m_5} = wm2B, тогда следующая конфигурация имеет вид

(time m_6) (inv I x_{13})(shc I x_2)(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_{13}\ \forall x_2\ Q_2(2m_6 + 1, 1 + x_{13}, 1 + x_2)$, где $m_6 < m_4 < m$, и он истинен по предположению индукции леммы 2. Шары $t_{m_5} \in \{$wm2C, wm2D, wh1$\}$ не приводят к развитию игры.

Если t_{m_5} = wh2, тогда необходимо x_2 = I x_{23} и следующая конфигурация имеет вид

[***]: (time m_6) (inv I I I x_{13})(shc x_{23})(shd I)(dirty)(exc).

[3] См. примечание, данное к шагу [12], шар t_{m_5} = rm2.

И мы снова видим конфигурацию похожую конфигурацию на одну из предыдущих, а именно на

[11]: (time m_5) (inv I I x_{13})(shc x_2)(shd I)(dirty)(exc).

Эти конфигурации описывают состояние игры в соответствующие моменты времени m_6 и $m_5 = m_6 + 1 > m_6$.

Лемма 3. *Игра Dragon начинающаяся с конфигурации* **[11]** *никогда не достигает конфигураций вида* (C_i)*, для всех* $1 \le i \le 4$*. Пусть* t *— время,* $2 + x_1$ *— содержимое корзины* **invalid***,* x_2 *— содержимое корзины* **shared_clean***, тогда обозначим утверждение леммы так* $\forall t\ \forall x_1\ \forall x_2\ Q_3(2t + 1, 2 + x_1, x_2)$.

Если наше доказательство рассматривать с точки появления конфигурации [11] ($t = m_5$), тогда оно является доказательством предиката $\forall\tau\ \forall x_1\ \forall x_2\ Q_3(2\tau + 1, 2 + x_1, x_2)$ индукцией по τ. Конфигурации [***] соответствует предикат $\forall x_{13}\ \forall x_{23}\ Q_3(2m_6 + 1, 3 + x_{13}, x_{23})$, где $m_6 < m_5 < m$, и он истинен по предположению индукции леммы 3. Шары $t_{m_5} \in \{$rm2C, rm2D, wm1$\}$ не приводят к развитию игры.

Если t_{m_5} = wh3, тогда следующая конфигурация имеет вид

[13]: (time m_6) (inv I I I x_{13})(shc x_2)(shd)(dirty)(exc).

Будем следить за [13], отложив разбор случая wh4 для шага [12].

Если $m_6 = 0$, то имеем базисный случай индукции ($m = 6$) и $\forall x_2$ Test($x_2, 0, 0, 0$) = True, иначе m_6 = I m_7, вынимаем шар t_{m_6} из корзины time, пометим этот шаг как [14] и перебираем значения t_{m_6}.

Шар t_{m_6} = rm1 необходимо требует $x_2 = 0$ и следующая конфигурация имеет вид

[*[4]]: (time m_7) (inv I I x_{13})(shc)(shd)(dirty)(exc I).

Эта конфигурация походит на конфигурацию

[3]: (time m_1) (inv x_1)(shc)(shd)(dirty)(exc I).

Конфигурации [*[4]], [3] описывают состояние игры в соответствующие моменты времени m_7 и $m_1 = m_7 + 6 > m_7$.

Лемма 4. *Игра Dragon начинающаяся с конфигурации* `[3]` *никогда не достигает конфигураций вида* (C_i)*, для всех* $1 \le i \le 4$*. Пусть* t *— время,* x_1 *— содержимое корзины* `invalid`*, тогда обозначим утверждение леммы так* $\forall t\ \forall x_1\ Q_4(2t+1, x_1)$.

Если наше доказательство рассматривать с точки появления конфигурации `[3]` ($t = m_1$), тогда оно является доказательством предиката $\forall\tau\ \forall x_1\ Q_4(2\tau+1, x_1)$ индукцией по τ. Конфигурации `[*`^{4}`]` соответствует предикат $\forall x_{13}\ Q_4(2m_7+1, 2+x_{13})$, где $m_7 < m_1 < m$, и он истинен по предположению индукции леммы 4.

Переходим к случаю t_{m_6} `= rm2`. Он требует x_2 `= I` x_{24} и следующая конфигурация:(time m_7)(inv I I x_{13})(shc I I x_{24})(shd)(dirty)(exc). Ей соответствует предикат $\forall x_{13}\ \forall x_{24}\ P(2m_7+1, 2+x_{13}, 2+x_{24})$, где $m_7 < m_2$, и он истинен по предположению индукции соответствующему утверждению, вынесенному в заголовок данного раздела 6.2.2; где m_2 — время начала доказательства этого утверждения.

Шар t_{m_6} `= wm1` необходимо требует $x_2 = 0$ и следующая конфигурация имеет вид

`[15]: (time` m_7`) (inv I I` x_{13}`)(shc )(shd )(dirty I)(exc ).`

Далее следим за развитием конфигурации `[15]`, отложив разбор случаев `wm2,` $\texttt{wh}_i$ $(1 \le i \le 4)$, оставшихся для шага `[14]`.

Если $m_7 = 0$, то имеем базисный случай индукции ($m = 7$) и `Test`$(0,0,1,0)$ `= True`, иначе m_7 `= I` m_8, вынимаем шар t_{m_7} из корзины `time`, пометим этот шаг как `[16]` и перебираем значения t_{m_7}. Шар t_{m_7} `= rm1` не приводит к развитию игры.

Если t_{m_7} `= rm2`, тогда следующая конфигурация имеет вид

`(time` m_8`) (inv I` x_{13}`)(shc I)(shd I)(dirty )(exc ).`

Ей соответствует предикат $\forall x_{13}\ Q_2(2m_8+1, 1+x_{13}, 1)$, где $m_8 < m_4 < m$, и он истинен по предположению индукции леммы 2. Шар t_{m_7} `= wm1` не приводит к развитию игры.

Если t_{m_7} `= wm2`, тогда следующая конфигурация имеет вид

`(time` m_8`) (inv I` x_{13}`)(shc I)(shd I)(dirty )(exc ).`

Как в случае `rm2`, он истинен по предположению индукции леммы 2.

Если t_{m_7} `= wh1`, тогда следующая конфигурация имеет вид

`(time` m_8`) (inv I I I` x_{13}`)(shc )(shd )(dirty )(exc ).`

Ей соответствует предикат $\forall x_{13}\ P(2m_8+1, 3+x_{13}, 0)$, где $m_8 < m$, и он истинен по предположению индукции, сделанному в самом начале нашего доказательства 6.2.1. Шары $t_{m_7} \in \{$`wh2, wh3, wh4`$\}$ не приводят к развитию игры.

Мы дошли до дна дерева разбора возможных случаев. Согласно нашему правилу разбора — depth first, мы должны рассмотреть оставшиеся случаи в порядке убывания номеров соответствующих шагов.

6.2.3. *Первый подъем по дереву разбора.* На шаге `[14]` мы отложили случаи $t_{m_6} \in \{\texttt{wm2}, \texttt{wh}_i\ \ (1 \le i \le 4)\}$. Рассмотрим их. Если t_{m_6} `= wm2`, тогда необходимо x_2 `= I` x_{25} и следующая конфигурация имеет вид

`(time` m_7`)(inv I I` x_{13}`)(shc I` x_{25}`)(shd I)(dirty)(exc).`

Ей соответствует предикат $\forall x_{13}\ \forall x_{25}\ Q_2(2m_7+1, 2+x_{13}, 1+x_{25})$, где $m_7 < m_4 < m$, и он истинен по предположению индукции леммы 2. Шар t_{m_6} `= wh1` не приводит к развитию игры.

Если t_{m_6} `= wh2`, тогда необходимо x_2 `= I` x_{26} и следующая конфигурация имеет вид

`[*`^{5}`]: (time` m_7`) (inv I I I I` x_{13}`)(shc` x_{26}`)(shd )(dirty )(exc ).`

Такую конфигурацию мы уже рассматривали, напомним ее

`[13]: (time` m_6`) (inv I I I` x_{13}`)(shc` x_2`)(shd )(dirty )(exc ).`

Конфигурации `[*`^{5}`], [13]` описывают состояние игры в соответствующие моменты времени m_7 и $m_6 = m_7 + 1 > m_7$.

Лемма 5. *Игра Dragon начинающаяся с конфигурации* `[13]` *никогда не достигает конфигураций вида* (C_i)*, для всех* $1 \le i \le 4$*. Пусть* t *— время,* $3 + x_1$ *— содержимое корзины* `invalid`*,* x_2 *— содержимое корзины* `shared_clean`*, тогда обозначим утверждение леммы так* $\forall t\ \forall x_1\ \forall x_2\ Q_5(2t+1, 3+x_1, x_2)$.

Если наше доказательство рассматривать с точки появления конфигурации `[13]` ($t = m_6$), тогда оно является доказательством предиката $\forall\tau\ \forall x_{13}\ \forall x_2\ Q_5(2\tau+1, 3+x_{13}, x_2)$ индукцией по τ. Конфигурации `[*`^{5}`]` соответствует предикат $\forall x_{13}\ \forall x_{26}\ Q_5(2m_7+1, 3+x_{13}, x_{26})$, где $m_7 < m_6 < m$, и он истинен по предположению индукции леммы 5.

Шары t_{m_6} = wh3,wh4 не приводят к развитию игры. Мы снова дошли до дна дерева разбора возможных случаев: по другой ветке.

6.2.4. *Второй подъем по дереву разбора.* На шаге [12] мы отложили случай t_{m_5} = wh4. Рассмотрим его. Прежде всего, заметим что, поднявшись по дереву выше конфигурации [13], мы завершили доказательство леммы 5[4].

Сам же случай t_{m_5} = wh4 тупиковый. И он опять последний вариант из всех возможных t_{m_5}. Продолжаем подниматься по ветке развития.

На шаге [10] мы отложили разбор случаев $t_{m_4} \in \{$wh3, wh4$\}$. Рассмотрим их. Мы поднялись по дереву выше конфигурации [11] и, следовательно, завершили доказательство леммы 3.

Если t_{m_4} = wh3, тогда следующая конфигурация имеет вид

[17]: (time m_5) (inv I I x_{13})(shc I x_2)(shd)(dirty)(exc).

Будем следить за [17], повторно отложив разбор случая wh4 для шага [10]. То есть опять опускаемся по новой ветке древа разбора.

Если $m_5 = 0$, то имеем еще один базисный случай индукции при $m = 5$; и $\forall x_2$ Test$(1 + x_2, 0, 0, 0)$ = True, иначе m_5 = I m_{6A}, вынимаем шар $t_{m_{5A}}$ из корзины time, пометим этот шаг как [18] и перебираем значения $t_{m_{5A}}$. Шар $t_{m_{5A}}$ = rm1 не приводит к развитию игры.

Если $t_{m_{5A}}$ = rm2, тогда следующая конфигурация имеет вид

(time m_{6A}) (inv I x_{13})(shc I I x_2)(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_{13}\ \forall x_2\ P(2m_{6A}+1, 1+x_{13}, 2+x_2)$, где $m_{6A} < m_2$, и он истинен по предположению индукции, сделанному в начале раздела 6.2.2. Шар $t_{m_{5A}}$ = wm1 тупиковый.

Если $t_{m_{5A}}$ = wm2, тогда следующая конфигурация имеет вид

(time m_{6A}) (inv I x_{13})(shc I x_2)(shd I)(dirty)(exc).

[4] При условии, что доказаны все утверждения от которых доказательство этой лемма зависит. Одно из них есть утверждение один нашей теоремы. И, следовательно, лемму 5 можно будет считать доказанной только при окончании всего нашего доказательства. Косвенным образом все сформулированные в доказательстве леммы также зависимы от первого утверждения теоремы. Мы позволим себе вольность, подобную указанной здесь, которая подчеркивает некоторые черты алгоритма доказательства.

Ей соответствует предикат $\forall x_{13}\ \forall x_2\ Q_2(2m_{6A}+1, 1+x_{13}, 1+x_2)$, где $m_{6A} < m_4 < m$, и он истинен по предположению индукции леммы 2. Шар $t_{m_{5A}}$ = wh1 не приводит к развитию игры.

Если $t_{m_{5A}}$ = wh2, тогда следующая конфигурация имеет вид

(time m_{6A}) (inv I I I x_{13})(shc x_2)(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_{13}\ \forall x_2\ Q_5(2m_7+1, 3+x_{13}, x_2)$ и он истинен по по лемме 5. Шары wh3, wh4 тупиковые. Это последние варианты из всех возможных $t_{m_{5A}}$. Мы находимся на дне третьей ветки развития.

6.2.5. *Третий подъем по дереву разбора.* На шаге [10] мы отложили (повторно — см. 6.2.4) случай t_{m_5} = wh4. Он тупиковый. Продолжаем подниматься по ветке. На шаге [8] мы отложили случаи $t_{m_3} \in \{$wh3, wh4$\}$. Рассмотрим их. Мы поднялись по дереву выше конфигурации [9] и, следовательно, завершили доказательство леммы 2.

Если t_{m_3} = wh3, тогда следующая конфигурация имеет вид

(time m_4) (inv I x_{13})(shc I I x_2)(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_{13}\ \forall x_2\ P(2m_4+1, 1+x_{13}, 2+x_2)$ (см. заголовок 6.2.2), где $m_4 < m_2$, и он истинен по предположению индукции, сделанному в начале раздела 6.2.2. Шар t_{m_3} = wh4 тупиковый. Мы находимся на дне четвертой ветки дерева развития.

6.2.6. *Четвертый подъем по дереву разбора.* На шаге [6] мы отложили случаи $t_{m_2} \in \{$wh$_i (1 \le i \le 4)\}$. Рассмотрим их. Мы поднялись по дереву выше конфигурации [7] и, следовательно, завершили доказательство леммы 1. Шар t_{m_2} = wh1 не приводит к развитию игры.

Если t_{m_2} = wh2, тогда следующая конфигурация имеет вид

[19]: (time m_3) (inv I x_1)(shc I x_2)(shd)(dirty)(exc).

Будем следить за развитием конфигурации [19], повторно отложив разбор случаев wh3, wh4 для шага [6]. То есть опять опускаемся по новой ветке древа разбора.

Если $m_3 = 0$, то имеем еще один базисный случай индукции при $m = 3$ и $\forall x_2$ `Test`$(1 + x_2, 0, 0, 0)$ `= True`, иначе m_3 `= I` m_{4B}, вынимаем шар $t_{m_{3B}}$ из корзины `time`, пометим этот шаг как `[20]` и перебираем значения $t_{m_{3B}}$. Шар $t_{m_{3B}}$ `= rm1` не приводит к развитию игры.

Если $t_{m_{3B}}$ `= rm2`, тогда следующая конфигурация имеет вид

`(time` m_{4B}`) (inv` x_1`)(shc I I` x_2`)(shd )(dirty )(exc ).`

Ей соответствует предикат $\forall x_1\ \forall x_2\ P(2m_{4B} + 1, x_1, 2 + x_2)$ (см. заголовок раздела 6.2.2), где $m_{4B} < m_2$, и он истинен по предположению индукции, сделанному в начале раздела 6.2.2. Шар $t_{m_{3B}}$ `=` `wm1` не приводит к развитию игры.

Если $t_{m_{3B}}$ `= wm2`, тогда следующая конфигурация имеет вид

`[21]: (time` m_{4B}`) (inv` x_1`)(shc I` x_2`)(shd I)(dirty )(exc ).`

Будем следить за развитием конфигурации `[21]`, отложив разбор случаев `wh`$_i$ $(1 \le i \le 4)$ для шага `[20]`.

Если $m_{4B} = 0$, то имеем базисный случай индукции $(m = 4B)$ и $\forall x_2$ `Test`$(1 + x_2, 1, 0, 0)$ `= True`, иначе m_{4B} `= I` m_{5B}, вынимаем шар $t_{m_{4B}}$ из корзины `time`, пометим этот шаг как `[22]` и перебираем значения $t_{m_{4B}}$. Шар $t_{m_{4B}}$ `= rm1` не приводит к развитию игры.

Если $t_{m_{4B}}$ `= rm2`, тогда необходимо x_1 `= I` x_{16} и следующая конфигурация:

`(time` m_{5B}`)(inv` x_{16}`)(shc I I` x_2`)(shd I)(dirty)(exc).`

Ей соответствует предикат $\forall x_{16}\ \forall x_2\ Q_1(2m_{5B}+1, x_{16}, 2+x_2)$, и он истинен по лемме 1. Шар $t_{m_{4B}}$ `= wm1` не приводит к развитию игры.

Если $t_{m_{4B}}$ `= wm2`, тогда необходимо x_1 `= I` x_{17} и следующая конфигурация:

`(time` m_{5B}`)(inv` x_{17}`)(shc I I` x_2`)(shd I)(dirty)(exc).`

И, как в случае `rm2`, соответствующий ей предикат истинен по лемме 1. Шар $t_{m_{4B}}$ `= wh1` не приводит к развитию игры.

Если $t_{m_{4B}}$ `= wh2`, тогда следующая конфигурация имеет вид

`[23]: (time` m_{5B}`) (inv I` x_1`)(shc` x_2`)(shd I)(dirty )(exc ).`

Будем следить за развитием конфигурации `[23]`, отложив разбор случаев `wh3, wh4` для шага `[22]`.

Если $m_{5B} = 0$, то имеем базисный случай индукции $(m = 5B)$ и $\forall x_2$ `Test`$(x_2, 1, 0, 0)$ `= True`, иначе m_{5B} `= I` m_{6B}, вынимаем шар $t_{m_{5B}}$ из корзины `time`, пометим этот шаг как `[24]` и перебираем значения $t_{m_{5B}}$. Шар $t_{m_{5B}}$ `= rm1` не приводит к развитию игры.

Случай $t_{m_{5B}}$ `= rm2` разобьем на подслучаи:

`(rm2A) shared_clean` ≥ 1`, (rm2B) shared_dirty` ≥ 1`,`
`(rm2C) dirty` ≥ 1`, (rm2D) exclusive` ≥ 1`.`

Если $t_{m_{5B}}$ `= rm2A`, тогда необходимо x_2 `= I` x_{27} и следующая конфигурация:

`(time` m_{6B}`)(inv` x_1`)(shc I I` x_{27}`)(shd I)(dirty)(exc).`

Ей соответствует предикат $\forall x_1\ \forall x_{27}\ Q_1(2m_{6B} + 1, x_1, 2 + x_{27})$, и он истинен по лемме 1.

Если $t_{m_{5B}}$ `= rm2B`, тогда следующая конфигурация имеет вид

`[*`^{6}`]: (time` m_{6B}`) (inv` x_1`)(shc I` x_2`)(shd I)(dirty )(exc ).`

Такую конфигурацию мы уже рассматривали, — это `[21]` со значением времени $m_{4B} = m_{6B} + 1 > m_{6B}$.

Лемма 6. *Игра Dragon начинающаяся с конфигурации `[21]` никогда не достигает конфигураций вида (C_i), для всех $1 \le i \le 4$. Пусть t — время, x_1 — содержимое корзины `invalid`, $1 + x_2$ — содержимое корзины `shared_clean`, тогда обозначим утверждение леммы так $\forall t\ \forall x_1\ \forall x_2\ Q_6(2t + 1, x_1, 1 + x_2)$.*

Если наше доказательство рассматривать с точки появления конфигурации `[21]` $(t = m_{4B})$, тогда оно является доказательством предиката $\forall \tau\ \forall x_1\ \forall x_2\ Q_5(2\tau + 1, x_1, 1 + x_2)$ индукцией по τ. Конфигурации `[*`^{6}`]` соответствует предикат $\forall x_1\ \forall x_2\ Q_6(2m_{6B} + 1, x_1, 1 + x_2)$, где $m_{6B} < m_{4B} < m$, и он истинен по предположению индукции леммы 6. Шары $t_{m_{5B}} \in \{$`rm2C, rm2D, wm1`$\}$ не приводят к развитию игры.

Случай $t_{m_{5B}}$ `= wm2` разобьем на подслучаи:

`(wm2A) shared_clean` ≥ 1`, (wm2B) shared_dirty` ≥ 1`,`
`(wm2C) dirty` ≥ 1`, (wm2D) exclusive` ≥ 1`.`

Если $t_{m_{5B}}$ `= wm2A`, тогда необходимо x_2 `= I` x_{28} и следующая конфигурация:

(time m_{6B})(inv x_1)(shc I I x_{28})(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ \forall x_{28}\ Q_1(2m_{6B}+1, x_1, 2+x_{28})$, и он истинен по лемме 1.

Если $t_{m_{5B}}$ = wm2B, тогда следующая конфигурация имеет вид:

(time m_{6B})(inv x_1)(shc I x_2)(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ \forall x_2\ Q_6(2m_{6B}+1, x_1, 1+x_2)$, где $m_{6B} < m_{4B} < m$, и он истинен по предположению индукции леммы 6. Шары $t_{m_{5B}} \in \{$wm2C, wm2D, wh1$\}$ не приводят к развитию игры.

Если $t_{m_{5B}}$ = wh2, тогда необходимо x_2 = I x_{29} и следующая конфигурация:

(time m_{6B})(inv I I x_1)(shc x_{29})(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ \forall x_{29}\ Q_3(2m_{6B}+1, 2+x_1, x_{29})$, и он истинен по лемме 3.

Если $t_{m_{5B}}$ = wh3, тогда следующая конфигурация имеет вид

[25]: (time m_{6B}) (inv I I x_1)(shc x_2)(shd)(dirty)(exc).

Будем следить за развитием [25], отложив случай wh4 для шага [24].

Если $m_{6B} = 0$, то имеем базисный случай индукции ($m = 6B$) и $\forall x_2$ Test$(x_2, 0, 0, 0)$ = True, иначе m_{6B} = I m_{7B}, вынимаем шар $t_{m_{6B}}$, пометим этот шаг как [26] и перебираем значения $t_{m_{6B}}$.

Если $t_{m_{6B}}$ = rm1, тогда необходимо $x_2 = 0$ и следующая конфигурация: (time m_{7B})(inv I x_1)(shc)(shd)(dirty)(exc I). Ей соответствует предикат $\forall x_1\ Q_4(2m_{7B}+1, 1+x_1)$, где $m_{7B} < m_1 < m$, и он истинен по предположению индукции леммы 4.

Если $t_{m_{6B}}$ = rm2, тогда необходимо x_2 = I x_{201} и следующая конфигурация:

(time m_{7B})(inv I x_1)(shc I I x_{201})(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ \forall x_{201}\ P(2m_{7B}+1, 1+x_1, 2+x_{201})$, где $m_{7B} < m_2$, и он истинен по предположению индукции, сделанному в начале раздела 6.2.2.

Если $t_{m_{6B}}$ = wm1, тогда необходимо $x_2 = 0$ и следующая конфигурация имеет вид

[27]: (time m_{7B}) (inv I x_1)(shc)(shd)(dirty I)(exc).

Будем следить за развитием конфигурации [27], отложив случаи wm2 и wh$_i$ $(1 \le i \le 4)$ для шага [26].

Если $m_{7B} = 0$, то имеем базисный случай индукции ($m = 7B$) и Test$(0, 0, 1, 0)$ = True, иначе m_{7B} = I m_{8B}, вынимаем шар $t_{m_{7B}}$ из корзины time, пометим этот шаг как [28] и перебираем значения $t_{m_{7B}}$. Шар $t_{m_{7B}}$ = rm1 не приводит к развитию игры.

Если $t_{m_{7B}}$ = rm2, тогда следующая конфигурация имеет вид

(time m_{8B}) (inv x_1)(shc I)(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ Q_6(2m_{8B}+1, x_1, 1)$, где $m_{8B} < m_{4B} < m$, и он истинен по предположению индукции леммы 6. Шар $t_{m_{7B}}$ = wm1 не приводит к развитию игры.

Если $t_{m_{7B}}$ = wm2, тогда следующая конфигурация имеет вид

(time m_{8B}) (inv x_1)(shc I)(shd I)(dirty)(exc).

Как и в случае $t_{m_{7B}}$ = rm2, доказательство рассматриваемого варианта шага [28] завершено по предположению индукции леммы 6.

Если $t_{m_{7B}}$ = wh1, тогда следующая конфигурация имеет вид

(time m_{8B}) (inv I I x_1)(shc)(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ P(2m_{8B}+1, 2+x_1, 0)$, где $m_{8B} < m$, и он истинен по предположению индукции, сделанному в самом начале нашего доказательства 6.2.1. Шары $t_{m_{7B}} \in$ {wh2, wh3, wh4} тупиковые. И мы находимся на дне пятой ветки дерева развития.

6.2.7. *Пятый подъем по дереву разбора.* На шаге [26] мы отложили случаи $t_{m_{6B}} \in \{$wm2, wh$_i$ $(1 \le i \le 4)\}$. Рассмотрим их. Если $t_{m_{6B}}$ = wm2, тогда необходимо x_2 = I x_{202} и следующая конфигурация: (time m_{7B})(inv I x_1)(shc I x_{202})(shd I)(dirty)(exc). Ей соответствует предикат $\forall x_1\ \forall x_{202}\ Q_2(2m_{7B}+1, 1+x_1, 1+x_{202})$, и он истинен по лемме 2. Шар $t_{m_{6B}}$ = wh1 тупиковый.

Если $t_{m_{6B}}$ = wh2, тогда необходимо x_2 = I x_{203} и следующая конфигурация: (time m_{7B})(inv I I I x_1)(shc x_{203})(shd)(dirty)(exc). Ей соответствует предикат $\forall x_1\ \forall x_{203}\ Q_5(2m_{7B}+1, 3+x_1, x_{203})$, и он истинен по лемме 5. Шары $t_{m_{6B}}$ = wh3, $t_{m_{6B}}$ = wh4 не приводят к развитию игры. Мы находимся на дне шестой ветки дерева развития.

6.2.8. *Шестой подъем по дереву разбора.* На шаге [24] мы отложили случай $t_{m_{5B}}$ = wh4. Он не приводит к развитию игры. Продолжаем подниматься по ветке. На шаге [22] мы отложили случаи $t_{m_{4B}} \in$ { wh3, wh4 }. Рассмотрим их.

Если $t_{m_{4B}}$ = wh3, тогда следующая конфигурация имеет вид

[*[7]]: (time m_{5B}) (inv I x_1)(shc I x_2)(shd)(dirty)(exc).

Такую конфигурацию мы уже рассматривали, — это [19] со значением времени $m_3 = m_{4B} + 1 > m_{4B}$.

Лемма 7. *Игра Dragon начинающаяся с конфигурации [19] никогда не достигает конфигураций вида* (C_i)*, для всех* $1 \le i \le 4$*. Пусть* t *— время,* $1 + x_1$ *— содержимое корзины* **invalid***,* $1 + x_2$ *— содержимое корзины* **shared_clean***, тогда обозначим утверждение леммы так* $\forall t\ \forall x_1\ \forall x_2\ Q_7(2t + 1, 1 + x_1, 1 + x_2)$.

Если наше доказательство рассматривать с точки появления конфигурации [19] ($t = m_3$), тогда оно является доказательством предиката $\forall \tau\ \forall x_1\ \forall x_2\ Q_7(2\tau + 1, 1 + x_1, 1 + x_2)$ индукцией по τ.

Конфигурации [*[7]] соответствует предикат $\forall x_1\ \forall x_2\ Q_7(2m_{4B} + 1, 1 + x_1, 1 + x_2)$, где $m_{4B} < m_3 < m$, и он истинен по предположению индукции леммы 7. Шар $t_{m_{4B}}$ = wh4, тупиковый. Мы снова дошли до дна дерева разбора возможных случаев: теперь уже по другой ветке. Мы находимся на дне седьмой ветки дерева развития.

6.2.9. *Седьмой подъем по дереву разбора.* На шаге [20] мы отложили случаи $t_{m_{3B}} \in \{\text{wh}_i\ \ (1 \le i \le 4)\}$. Рассмотрим их. Мы поднялись по дереву выше конфигурации [21] и, следовательно, завершили доказательство леммы 7. Шар $t_{m_{3B}}$ = wh1 тупиковый.

Если $t_{m_{3B}}$ = wh2, тогда следующая конфигурация имеет вид

[29]: (time m_{4B}) (inv I I x_1)(shc x_2)(shd)(dirty)(exc).

Следим за [29], повторно отложив разбор случаев wh3, wh4 для шага [20]. То есть опять опускаемся по новой ветке древа разбора.

Если $m_{4B} = 0$, то имеем базисный случай индукции ($m = 4B$) и $\forall x_2$ Test($x_2, 0, 0, 0$) = True, иначе m_{4B} = I m_{5C}, вынимаем шар $t_{m_{4C}}$, пометим этот шаг как [30] и перебираем значения $t_{m_{4C}}$.

Если $t_{m_{4C}}$ = rm1, тогда необходимо $x_2 = 0$ и следующая конфигурация: (time m_{5C})(inv I x_1)(shc)(shd)(dirty)(exc I). Ей соответствует предикат $\forall x_1\ Q_4(2m_{5C} + 1, 1 + x_1)$, где $m_{5C} < m_1 < m$, и он истинен по предположению индукции леммы 4.

Если $t_{m_{4C}}$ = rm2, тогда необходимо x_2 = I x_{204} и следующая конфигурация:

(time m_{5C})(inv I x_1)(shc I I x_{204})(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ \forall x_{204}\ P(2m_{5C} + 1, 1 + x_1, 2 + x_{204})$, где $m_{5C} < m_2$, и он истинен по предположению индукции, сделанному в начале раздела 6.2.2.

Если $t_{m_{4C}}$ = wm1, тогда необходимо $x_2 = 0$ и следующая конфигурация имеет вид (time m_{5C})(inv I x_1)(shc)(shd)(dirty I)(exc). Мы ее рассматривали под номером [27] на другой ветке развития (см. 6.2.6). Доказательство случая $t_{m_{4C}}$ = wm1 шага [30] закончено[5].

Если $t_{m_{4C}}$ = wm2, тогда необходимо x_2 = I x_{205} и следующая конфигурация:

(time m_{5C})(inv I x_1)(shc I x_{205})(shd I)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ \forall x_{205}\ Q_2(2m_{5C} + 1, 1 + x_1, 1 + x_{205})$, и он истинен по лемме 2. Шар $t_{m_{4C}}$ = wh1 тупиковый.

Если $t_{m_{4C}}$ = wh2, тогда необходимо x_2 = I x_{206} и следующая конфигурация:

(time m_{5C})(inv I I I x_1)(shc x_{206})(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ \forall x_{206}\ Q_5(2m_{5C} + 1, 3 + x_1, x_{206})$ и он истинен по лемме 5. Шары $t_{m_{4C}} \in \{$wh3, wh4$\}$ не приводят к развитию игры. Мы находимся на дне восьмой ветки дерева развития.

[5] Отметим, что при реальной работе суперкомпилятор SCP4 перевычисляет все указанные случаи заново.

6.2.10. *Восьмой подъем по дереву разбора.* На шаге [6] мы повторно отложили случаи $t_{m_2} \in \{\texttt{wh3, wh4}\}$. Они не приводят к развитию игры. Продолжаем подниматься по ветке. На шаге [4] мы отложили случаи $t_{m_1} \in \{\texttt{wm1, wm2, wh}_i (1 \le i \le 4)\}$. Рассмотрим их. Мы поднялись по дереву выше конфигураций [19], [5g] и, следовательно, завершили доказательство леммы 7 и утверждения сформулированного в заголовке 6.2.2. Шар t_{m_1} = wh1 тупиковый.

Если t_{m_1} = wm2, тогда необходимо x_1 = I x_{18} и следующая конфигурация: (time m_2)(inv x_{18})(shc I)(shd I)(dirty)(exc). Ей соответствует предикат $\forall x_{18}\ Q_6(2m_2+1, x_{18}, 1)$, и он истинен по лемме 6. Шары $t_{m_1} \in \{\texttt{wh1, wh2, wh3}\}$ не приводят к развитию игры.

Если t_{m_1} = wh4, тогда следующая конфигурация имеет вид

(time m_2) (inv I x_1)(shc)(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ P(2m_2+1, 1+x_1, 0)$, где $m_2 < m$, и он истинен по предположению индукции, сделанному в самом начале нашего доказательства 6.2.1. Мы находимся на дне девятой ветки дерева развития.

6.2.11. *Девятый подъем по дереву разбора.* На шаге [2] мы отложили случаи $t_m \in \{\texttt{rm2, wm1, wm2, wh}_i (1 \le i \le 4)\}$. Рассмотрим их. Мы поднялись по дереву выше конфигурации [3] и, следовательно, завершили доказательство леммы 4. Шар t_m = rm2 тупиковый.

Если t_m = wm1, тогда и следующая конфигурация имеет вид

[31]: (time m_1) (inv x_1)(shc)(shd)(dirty I)(exc).

Будем следить за развитием конфигурации [31], отложив повторно разбор случаев wm2 и $\texttt{wh}_i$ $(1 \le i \le 4)$ для шага [2]. Опять опускаемся по новой ветке древа разбора.

Если $m_1 = 0$, то имеем базисный случай индукции ($m = 1$) и Test$(0,0,1,0)$ = True, иначе m_1 = I m_{2D}, вынимаем шар $t_{m_{1D}}$ из корзины time, пометим этот шаг как [32] и перебираем значения $t_{m_{1D}}$. Шар $t_{m_{1D}}$ = rm1 не приводит к развитию игры.

Если $t_{m_{1D}}$ = rm2, тогда необходимо x_1 = I x_{19} и следующая конфигурация: (time m_{2D})(inv x_{19})(shc I)(shd I)(dirty)(exc). Ей соответствует предикат $\forall x_{19}\ Q_6(2m_{2D}+1, x_{19}, 1)$, и он истинен по лемме 6. Шар $t_{m_{1D}}$ = wm1 не приводит к развитию игры.

Если $t_{m_{1D}}$ = wm2, тогда необходимо x_1 = I x_{101} и следующая конфигурация:

(time m_{2D})(inv x_{101})(shc I)(shd I)(dirty)(exc).

Как и в случае rm2, ей соответствующий предикат истинен по лемме 6.

Если $t_{m_{1D}}$ = wh1, тогда следующая конфигурация имеет вид

(time m_{2D}) (inv I x_1)(shc)(shd)(dirty)(exc).

Ей соответствует предикат $\forall x_1\ P(2m_{2D}+1, 1+x_1, 0)$, где $m_{2D} < m$, и он истинен по предположению индукции, сделанному в самом начале нашего доказательства 6.2.1. Шары $t_{m_{1D}} \in \{\texttt{wh2, wh3, wh4}\}$ тупиковые. Мы находимся на дне десятой ветки дерева развития.

6.2.12. *Десятый подъем по дереву разбора.* На шаге [2] мы повторно отложили случаи $t_m \in \{\texttt{wm2, wh}_i\ (1 \le i \le 4)\}$. Они тупиковые. Продолжаем подниматься по ветке. Мы на вершине нашего дерева доказательства! Теорема, сформулированная в разделе 6.1, доказана.

6.3. Замечания к доказательству. Индукция ведется по одной переменной — времени, чисто техническую трудность представляет перебор многих случаев: что естественно поручить алгоритму[6]. Большую часть этого алгоритма мы и продемонстрировали выше. Неалгоритмизированным остался лишь шаг формулировки самой теоремы, то есть построения конфигурации $[I_2]$. Почему мы решили построить $[I_2]$, и как мы ее построили? Это критический момент. За построение правильной формулировки теоремы отвечает алгоритм обобщения конфигураций, используемый суперкомпилятором SCP4. Ограничения на объем статьи вынуждают нас отослать читателя к работе [1], где обсуждаются некоторые принципы работы этого алгоритма. Более детальное описание актуальной версии SCP4 можно найти в [8, 9]. Отметим, что формулировки утверждений лемм возникли естественно в ходе доказательства, и нам нужно было только обнаружить (просмотром уже проведенной части доказательства), что текущая конфигурация $[*^i]$ есть частный случай некоторой ранее рассмотренной конфигурации $[\texttt{L}_i]$.

[6] Как мы, смеем надеяться, убедили читателя.

7. Результат преобразований

Скелет обобщенных утверждений нашей теоремы, которые суперкомпилятор сумел доказать представлены в остаточной программе:

```
* InputFormat: <Go e.t (e.x1)>
$ENTRY Go {
 e.t (e.x1)  = <F5 (e.t) e.x1> ;
}

* InputFormat: <F185 (e.t) (e.x1) e.x2>
F185 {
 () (e.x1) e.x2  = True ;
 (rm2A e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (rm2B e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wm2A e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wm2B e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wh2 ) (e.x1) e.x2  = True ;
 (wh2 rm2A e.t) (e.x1) I e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wh2 rm2B e.t) (e.x1) e.x2  = <F185 (e.t) (e.x1) e.x2> ;
 (wh2 wm2A e.t) (e.x1) I e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wh2 wm2B e.t) (e.x1) e.x2  = <F185 (e.t) (e.x1) e.x2> ;
 (wh2 wh2 e.t) (e.x1) I e.x2  = <F89 (e.t) (e.x1) e.x2> ;
 (wh2 s.t2) (e.x1) e.x2  = True ;
 (wh2 s.t2 rm1 e.t) (e.x1)  = <F12 (e.t) I e.x1> ;
 (wh2 s.t2 rm2A e.t) (e.x1) I e.x2  = <F32 (e.t) (I e.x1) e.x2> ;
 (wh2 s.t2 wm1 ) (e.x1)  = True ;
 (wh2 s.t2 wm1 rm2C e.t) (e.x1)  = <F185 (e.t) (e.x1)> ;
 (wh2 s.t2 wm1 wm2C e.t) (e.x1)  = <F185 (e.t) (e.x1)> ;
 (wh2 s.t2 wm1 s.t4 e.t) (e.x1)  = <F5 (e.t) I e.x1> ;
 (wh2 s.t2 wm2A e.t) (e.x1) I e.x2  = <F66 (e.t) (e.x1) e.x2> ;
 (wh2 s.t2 s.t3 e.t) (e.x1) I e.x2  = <F115 (e.t) (e.x1) e.x2> ;
 (s.t1 e.t) (e.x1) e.x2  = <F174 (e.t) (e.x1) e.x2> ;
}

* InputFormat: <F174 (e.t) (e.x1) e.x2>
F174 {
 () (e.x1) e.x2  = True ;
 (rm2A e.t) (e.x1) e.x2  = <F32 (e.t) (e.x1) e.x2> ;
 (wm2A e.t) (e.x1) e.x2  = <F185 (e.t) (e.x1) e.x2> ;
 (s.t1) (e.x1) e.x2  = True ;
 (s.t1 rm1 e.t) (e.x1)  = <F12 (e.t) I e.x1> ;
 (s.t1 rm2A e.t) (e.x1) I e.x2  = <F32 (e.t) (I e.x1) e.x2> ;
 (s.t1 wm1) (e.x1)  = True ;
```



```
 (s.t1 wm1 rm2C e.t) (e.x1)  = <F185 (e.t) (e.x1)> ;
 (s.t1 wm1 wm2C e.t) (e.x1)  = <F185 (e.t) (e.x1)> ;
 (s.t1 wm1 s.t3 e.t) (e.x1)  = <F5 (e.t) I e.x1> ;
 (s.t1 wm2A e.t) (e.x1) I e.x2  = <F66 (e.t) (e.x1) e.x2> ;
 (s.t1 s.t2 e.t) (e.x1) I e.x2  = <F115 (e.t) (e.x1) e.x2> ;
}

* InputFormat: <F115 (e.t) (e.x1) e.x2>
F115 {
 () (e.x1) e.x2  = True ;
 (rm1 e.t) (e.x1)  = <F12 (e.t) I I e.x1> ;
 (rm2A e.t) (e.x1) I e.x2  = <F32 (e.t) (I I e.x1) e.x2> ;
 (wm1 ) (e.x1)  = True ;
 (wm1 rm2C e.t) (e.x1)  = <F66 (e.t) (e.x1)> ;
 (wm1 wm2C e.t) (e.x1)  = <F66 (e.t) (e.x1)> ;
 (wm1 s.t2 e.t) (e.x1)  = <F5 (e.t) I I e.x1> ;
 (wm2A e.t) (e.x1) I e.x2  = <F66 (e.t) (I e.x1) e.x2> ;
 (s.t1 e.t) (e.x1) I e.x2  = <F115 (e.t) (I e.x1) e.x2> ;
}

* InputFormat: <F89 (e.t) (e.x1) e.x2>
F89 {
 () (e.x1) e.x2  = True ;
 (rm2A e.t) (e.x1) I e.x2  = <F43 (e.t) (I e.x1) e.x2> ;
 (rm2B e.t) (e.x1) e.x2  = <F66 (e.t) (e.x1) e.x2> ;
 (wm2A e.t) (e.x1) I e.x2  = <F43 (e.t) (I e.x1) e.x2> ;
 (wm2B e.t) (e.x1) e.x2  = <F66 (e.t) (e.x1) e.x2> ;
 (wh2 e.t) (e.x1) I e.x2  = <F89 (e.t) (I e.x1) e.x2> ;
 (s.t1 e.t) (e.x1) e.x2  = <F115 (e.t) (e.x1) e.x2> ;
}

* InputFormat: <F66 (e.t) (e.x1) e.x2>
F66 {
 () (e.x1) e.x2  = True ;
 (rm2A e.t) (e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (rm2B e.t) (e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wm2A e.t) (e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wm2B e.t) (e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (wh2 e.t) (e.x1) e.x2  = <F89 (e.t) (e.x1) e.x2> ;
 (s.t1) (e.x1) e.x2  = True ;
 (s.t1 rm2A e.t) (e.x1) e.x2  = <F32 (e.t) (I e.x1) e.x2> ;
 (s.t1 wm2A e.t) (e.x1) e.x2  = <F66 (e.t) (e.x1) e.x2> ;
 (s.t1 s.t2 e.t) (e.x1) e.x2  = <F115 (e.t) (e.x1) e.x2> ;
```

```
}

* InputFormat: <F43 (e.t) (e.x1) e.x2>
F43 {
 () (e.x1) e.x2  = True ;
 (rm2A e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) I e.x2> ;
 (rm2B e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) I e.x2> ;
 (wm2A e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) I e.x2> ;
 (wm2B e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) I e.x2> ;
 (wh2 e.t) (e.x1) e.x2  = <F66 (e.t) (e.x1) e.x2> ;
 (s.t1 e.t) (e.x1) e.x2  = <F32 (e.t) (I e.x1) e.x2> ;
}

* InputFormat: <F32 (e.t) (e.x1) e.x2>
F32 {
 () (e.x1) e.x2  = True ;
 (rm2A e.t) (I e.x1) e.x2  = <F32 (e.t) (e.x1) I e.x2> ;
 (wm2A e.t) (I e.x1) e.x2  = <F43 (e.t) (e.x1) e.x2> ;
 (s.t1 e.t) (e.x1) e.x2  = <F174 (e.t) (e.x1) e.x2> ;
}

* InputFormat: <F12 (e.t) e.101 >
F12 {
 () e.101  = True ;
 (rm2D e.t) I e.x1  = <F32 (e.t) (e.x1)> ;
 (wm2D e.t) I e.x1  = <F185 (e.t) (e.x1)> ;
 (s.t1 e.t) e.x1  = <F5 (e.t) e.x1> ;
}

* InputFormat: <F5 (e.t) e.x1>
F5 {
 () e.x1  = True ;
 (rm1 e.t) e.x1  = <F12 (e.t) e.x1> ;
 (s.t1) e.x1  = True ;
 (s.t1 rm2C e.t) I e.x1  = <F185 (e.t) (e.x1)> ;
 (s.t1 wm2C e.t) I e.x1  = <F185 (e.t) (e.x1)> ;
 (s.t1 s.t2 e.t) e.x1  = <F5 (e.t) e.x1> ;
}
```

Где мы изменили только имена переменных — для лучшей читабельности текста. Утверждения нашей теоремы и леммы закодированы во входных форматах рекурсивных функций. Соответствие номерам лемм: Л1–`F43`, Л2–`F66`, Л3–`F89`, Л4–`F12`, Л5–`F115`, Л6–`F185`, Л7–`F174`. Первое утверждение теоремы соответствует `F5`, второе — `F32`.

Формальное *синтаксическое* свойство остаточной программы, на основании которого мы можем заключить, что закодированный посредством входной программы протокол корректен по данному постусловию, — отсутствие выходов из рекурсий (пассивных правых частей предложений) с результатом `False`.

8. Заключение

Пседокомментарии в программе

```
*$MST_FROM_ENTRY;
*$STRATEGY Applicative;
*$LENGTH 0;
```

являются настройками вызова суперкомпилятора: первый предоставляет синтаксические удобства пользователю; второй указывает на то, что в момент мета–интерпретации функции будут вызываться по значению (by value); третий сокращает количество разверток в момент мета–интерпретации до минимума предусмотренного в SCP4, уменьшая время суперкомпиляции. В случае протокола Dragon, основная стадия работы SCP4, после которой уже можно сделать заключение о корректности протокола, занимает 6 секунд (Windows XP/Service Pack 2, Intel Pentium III, 450 MHz, 256 MB of RAM), последующие стадии работы преобразователя SCP4 (см. [9, 10]), занимают существенно большее время. Время всей сессии SCP4 на данной задаче равно 3 минутам.

Мы показали, что достаточно универсальный инструмент — оптимизатор SCP4 может быть использован для автоматической верификации программ, если саму верификацию мы будем понимать как параметризованное тестирование.

Список литературы

[1] Лисица А. П., Немытых А. П. *Верификация как параметризованное тестирование (эксперименты с суперкомпилятором SCP4).* (Представлено для публикации в журнале «Программирование»).

[2] Олехник С. Н., Нестеренко Ю. В., Потапов М. К. Старинные занимательные задачи. — Москва: Наука, 1988.

[3] Bundy A. *The Automation of Proof by Mathematical Induction.* // Handbook of Automated Reasoning., 2001, с. 845–911.

[4] Delzanno G. *Automatic Verification of Parameterized Cache Coherence Protocols* // Of the 12th Int. Conference on Computer Aided Verification: LNCS. — Т. **1855**: Springer–Verlag, 2000, с. 53–68.

[5] Delzanno G. Automatic Verification of Cache Coherence Protocols via Infinite-state Constraint-based Model Checking, `http://www.disi.unige.it/person/DelzannoG/protocol.html`.

[6] Delzanno G. *Verification of Consistency Protocols via Infinite–state Symbolic Model Checking, A Case Study* // Of FORTE/PSTV, 2000, с. 171–188.

[7] Lisitsa A. P., Nemytykh A. P. *Verification of parameterized systems using supercompilation. A case study* // Of APPSEM05 ред. Hofmann M., Loidl H. W. — Germany, 2005, Accessible via: `ftp://www.botik.ru/pub/local/scp/refal5/appsem_verification2005.ps`.

[8] Nemytykh A. P. *The Supercompiler SCP4: General Structure (extended abstract)* // Of the Perspectives of System Informatics: LNCS. — Т. **2890**: Springer–Verlag, 2003, с. 162–170, Accessible via:`ftp://www.botik.ru/pub/local/scp/refal5/nemytykh_PSI03.ps.gz`.

[9] Nemytykh A. P. *The Supercompiler SCP4: General Structure*: Программные системы: теория и применение. — Т. **1**. — Москва: Физматлит, 2004, с. 448–485, Available at `ftp://ftp.botik.ru/pub/local/scp/refal5/GenStruct.ps.gz`.

[10] Nemytykh A. P., Turchin V. F. The Supercompiler SCP4: sources, on–line demonstration, 2000, `http://www.botik.ru/pub/local/scp/refal5/`.

[11] Turchin V. F. *The concept of a supercompiler*: ACM Transactions on Programming Languages and Systems. — Т. **8**: ACM Press, 1986, с. 292–325.

[12] Turchin V. F. Refal–5, Programming Guide and Reference Manual. — Holyoke, Massachusetts: New England Publishing Co., 1989, (electronic version: `http://www.botik.ru/pub/local/scp/refal5/` ,2000).

[13] Turchin V. F., Turchin D. V., Konyshev A. P., Nemytykh A. P. Refal–5: sources, executable modules, 2000, `http://www.botik.ru/pub/local/scp/refal5/`.

The University of Liverpool

Исследовательский центр мультипроцессорных систем ИПС РАН

A. P. Lisitsa, A. P. Nemytykh. *Work on errors.* (in Russian.)

Abstract. Errors in publications are a natural product of human life. Some errors lead to more substantial positive consequences than their absence. We consider two such errors found automatically by a specializer SCP4 of functional programs. We start with a simple example and finish with a successful verification of a cache coherence protocol; id est SCP4 detects an error in a description of the Xerox PARC Dragon given in [4] and automatically proves robustness of another description of the protocol.