Модел. и анализ информ. систем. Т. **19**, № **6** (2012) 57–68

УДК 519.7+004.75

Дедуктивная верификация протокола скользящего окна

Шкляев Д.А., Непомнящий В.А.

Институт систем информатики имени А.П. Ершова СО РАН

e-mail: dmitrichkl@yahoo.com, vnep@iis.nsk.su

получена 22 июля 2012

Ключевые слова: коммуникационные протоколы, протокол скользящего окна, отказоустойчивость, формальная спецификация, автоматизированная верификация, интерактивное доказательство теорем, PVS

Рассматривается известный протокол скользящего окна, который обеспечивает надёжную и эффективную передачу данных по ненадёжным каналам. Формальное доказательство корректности этого протокола требует преодоления существенных трудностей, связанных с высокой степенью параллелизма, которая создаёт значительные возможности для ошибок. Здесь рассматривается версия данного протокола, основанная на выборочном повторе кадров. На языке системы верификации PVS описаны спецификация этого протокола с помощью машины состояний и его свойство безопасности. С помощью системы PVS проведено в интерактивном режиме доказательство этого свойства протокола скользящего окна.

1. Введение

Одним из известных протоколов для надёжной передачи данных по ненадёжным каналам является протокол скользящего окна (Sliding Window Protocol, SWP) [9, 2]. На нём основаны практически важные коммуникационные протоколы, такие как TCP и HDLC.

Корректность протокола SWP далеко не очевидна, потому что для него характерно тонкое взаимодействие нескольких распределённых компонент, а степень параллелизма в нём весьма высока. Поэтому были попытки верифицировать этот протокол формально, представляя его на некотором языке спецификаций. Эти попытки верификации нередко были поддержаны методом проверки моделей или интерактивным доказательством теорем. Однако оказалось, что формальная верификация SWP является трудной задачей, и к настоящему времени ещё не достигнут консенсус по поводу того, какие формальные модели и методы наиболее подходят для

этого. Обсуждение общих вопросов, связанных с математическим моделированием SWP и других коммуникационных протоколов, можно найти в [2] и [1].

В диссертации [4] был ранее представлен общий метод для спецификации и верификации протоколов управления базами данных, который был применён к верификации нескольких нетривиальных примеров из данной области. Естественно возникает задача обобщения этого метода для верификации протоколов скользящего окна. Используя модификацию нашего метода, в работе [5], мы уже верифицировали версию SWP, в которой для удаления «старых» сообщений из каналов использовался необычный временной механизм, что позволяло переупорядочивать сообщения в каналах. Все доказательства в [4] и [5] были автоматизированы с помощью системы верификации PVS [7].

Цель данной работы – рассмотреть стандартную версию протокола скользящего окна, предназначенную для каналов, не допускающих переупорядочивания сообщений. Именно эта версия обычно рассматривается в литературе. Она соответствует протоколу с выборочным повтором кадров из [2], то есть довольно сложному варианту SWP, в котором окна отправителя и получателя имеют произвольный размер. В данной работе описана формальная спецификация и верификация с помощью системы PVS этой версии протокола SWP. Заметим, что данная работа существенно обобщает результаты нашей работы [6], в которой размер окна получения сообщений был ограничен 1.

Данная статья организована следующим образом. В разделе 2 даётся неформальное описание данной версии протокола скользящего окна. В разделе 3 мы формализуем данный протокол с помощью машины состояний. В разделе 4 изложена спецификация и верификация свойства безопасности для этого протокола. В разделе 5 содержится краткий обзор родственных работ.

2. Протокол скользящего окна: неформальное описание

Отправитель и получатель. В протоколе скользящего окна имеются две основные компоненты: отправитель и получатель. Отправитель получает бесконечную последовательность данных от *посылающего узла*, принадлежащего к более высокому по отношению к протоколу *сетевому уровню*. Мы называем неделимые сегменты данных в этой последовательности *кадрами*, а саму последовательность *входной последовательностью*. Входная последовательность должна быть передана получателю по ненадёжной сети. Получив кадр через канал связи, получатель может решить принять этот кадр и через некоторое время доставить его *принимающему узлу* (также принадлежащему к сетевому уровню). Основное условие корректности для протокола гласит, что получатель должен доставлять кадры принимающему узлу в том же порядке, в каком они появляются во входной последовательности.

Сообщения и каналы. Чтобы передать кадр, отправитель помещает его в *сообщение с кадром* вместе с некоторой дополнительной информацией и посылает в канал кадров. После того, как получатель через некоторое время принимает сообщение с кадром из канала, он обратно посылает отправителю *сообщение с под-*

тверждением, подтверждающее соответствующий кадр. Это сообщение передаётся через *канал подтверждений*. После получения сообщения с подтверждением отправитель может быть уверен, что соответствующий кадр был получен получателем.

Порядковые номера. Отправитель посылает кадры в том же порядке, в каком они встречаются во входной последовательности. Однако канал кадров является ненадёжным, и поэтому получатель может получить эти кадры в ином порядке (если вообще получит). Отсюда понятно, что каждое сообщение с кадром должно содержать некоторую информацию о порядке соответствующего кадра во входной последовательности. Такая дополнительная информация называется *порядковым номером*. Если рассматриваемая система не налагает каких-либо ограничений на размер сообщения с кадром, то мы можем просто посылать с каждым кадром его порядковый номер во входной последовательности (например, 0, 1, 2, 3 и т.д.). Именно такую ситуацию мы рассматриваем в данной работе. Таким образом, данная версия протокола использует *неограниченное* число порядковых номеров.

Для подтверждения кадра, в сообщении с подтверждением получатель указывает порядковый номер, с которым кадр был получен. Подтверждения являются «накопительными»: например, когда получатель подтверждает кадр с порядковым номером 3, это значит, что кадры с порядковыми номерами 0, 1 и 2 уже были приняты.

Окно отправителя. В любой момент времени отправитель запоминает последовательность порядковых номеров, соответствующих кадрам, которые ему разрешается послать. Говорят, что эти кадры принадлежат к *окну отправителя*. Аналогично, получатель имеет *окно получателя* с порядковыми номерами, которые ему разрешается принять. Во многих версиях протокола, включая [2], размеры окон отправителя и получателя равны. Здесь мы обозначаем размеры окон отправителя и получателя как SW и RW соответственно, не предполагая, что SW = RW, что приводит к более общему протоколу скользящего окна.

Во время исполнения протокола возможна ситуация, когда некоторые кадры в начале окна отправителя уже были посланы, но ещё не подтверждены, а остальные кадры ещё не были посланы. Когда прибывает подтверждение для уже посланного кадра в окне отправителя, этот кадр и все предшествующие кадры удаляются из окна (из-за накопительного характера подтверждений). Одновременно с этим окно «сдвигается вперёд», так что оно вновь содержит не более SW кадров. В результате отправитель может послать дополнительные кадры. Подтверждения для кадров, не входящих в окно, сразу выбрасываются.

Если подтверждение для посланного кадра долго не приходит, обычно это означает, что был потерян либо сам кадр, либо подтверждение для него. Чтобы обеспечить продвижение протокола, такой кадр *пересылают* (посылают снова). Согласно [2], существует много разных стратегий для посылки и пересылки кадров, принимающих во внимание, например, эффективное распределение ресурсов и необходимость избежать «засорения» сети. В данной работе мы абстрагируемся от таких деталей посылки и пересылки кадров и специфицируем только такие ограничения на поведение протокола, которые необходимы для обеспечения его свойства безопасности.

Окно получателя. Во время исполнения в окне получателя обычно череду-

ются порядковые номера, соответствующие кадрам, принятым не в соответствии с порядком их посылки, и порядковые номера, соответствующие «пустым местам», т.е. кадрам, прибытие которых ещё ожидается. Когда прибывает кадр с порядковым номером, соответствующим некоторому «пустому месту», он принимается, т.е. вставляется в окно, а иначе выбрасывается. В любой момент времени, если первым элементом окна получателя является кадр, он может быть доставлен принимающему узлу, а окно при этом сдвигается на один элемент.

Порядковый номер последнего доставленного кадра может быть послан обратно отправителю для подтверждения этого кадра (для упрощения спецификации, в нашей версии протокола мы подтверждаем не принятые, а уже доставленные кадры). Не обязательно подтверждать каждый кадр: можно доставить несколько кадров подряд, а затем подтвердить только последний из них. Если получатель длительное время не имеет новых кадров для доставки, он должен переслать последнее подтверждение для обеспечения продвижения протокола.

3. Протокол скользящего окна: спецификация в PVS

В нашем методе протокол SWP представляется абстрактной машиной состояний, допускающей бесконечное число состояний и переходов между ними. Желаемые свойства протокола формализуются как логические формулы на всех трассах состояний и операций (переходов), порождаемых машиной состояний протокола.

Более формально, в нашем методе протокол определяется понятием *состояния*, представляющим текущие значения переменных во время исполнения протокола, и множеством *операций*. Состояние состоит из переменных для данных и контрольных переменных (которые у нас формально не различаются), каждая из которых принадлежит одной из распределённых компонент. Например, в протоколе скользящего окна имеются четыре компоненты: отправитель, получатель, канал кадров и канал подтверждений. Каждая операция исполняется на одной из этих распределённых компонент и меняет её переменные, а также, возможно, переменные соседней компоненты, с которой происходит интеракция. Операции, которые могут иметь произвольное число параметров, специфицируются предусловием и оператором эффекта, связывающим состояния до и после исполнения операции. Исполнение протокола, или *пробег*, представляется бесконечной последовательностью вида $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ...s_i \xrightarrow{a_i} s_{i+1} \xrightarrow{a_{i+1}} ...$, где s_i – это состояния, a_i – это выполненные операции, s_0 – это начальное состояние, каждое s_i удовлетворяет предусловию a_i, и каждая пара (s_i, s_{i+1}) соответствует эффекту a_i.

В спецификации PVS, для реализации этого определения пробегов необходимо дать начальное состояние Ini, предусловие Pre, то есть булевский предикат на парах (s_i, a_i), а также предикат эффекта Eff, то есть булевский предикат на тройках (s_i, a_i, s_{i+1}). После этого пробег r определяется как пара, состоящая из бесконечной последовательности состояний $st(r)$ и бесконечной последовательности операций $act(r)$ и удовлетворяющая трём следующим свойствам:

1. $st(r)(0) = Ini$;

2. для каждого натурального индекса i выполняется $Pre(st(r)(i), act(r)(i)) = true$;

3. для каждого натурального индекса i, выполняется $Eff(st(r)(i), act(r)(i), st(r)(i+1)) = true$.

Чтобы специфицировать протокол скользящего окна, мы должны определить структуру состояний и операций в данном протоколе. Чтобы получить исполнения протокола, мы также должны заменить начальное состояние Ini и предикаты Pre и Eff их значениями $SWIni$, $SWPre$ и $SWEff$, определёнными специально для SWP.

В протоколе скользящего окна можно по-разному моделировать отправителя и получателя. В нашей модели отправителя его окно «скользит» по бесконечной входной последовательности *input*. Мы не специфицируем структуру отдельных кадров во входной последовательности. Переменная $first$ указывает на первый кадр в окне отправителя, переменная $ftsend$ обозначает первый кадр, который ещё не был послан, причём всегда должно выполняться $first \leq ftsend \leq first + N$. Таким образом, в любой момент времени, кадры с индексами от $first$ до $ftsend - 1$ (если такие есть) уже были посланы, но ещё не были подтверждены, а кадры с индексами от $ftsend$ до $first + N - 1$ (если такие есть) находятся в окне отправителя, но ещё не были посланы. Полная структура данных для отправителя выглядит следующим образом.

Sender:

1) $input : sequence[Frames]$,
2) $first : nat$,
3) $ftsend : nat$

Для получателя, *output* – это конечная выходная последовательность, *rwindow* – это окно получателя, имеющее в точности RW элементов (которые могут быть либо кадрами, либо пустыми элементами, обозначаемыми как ε), *ackseqnum* – это последний доставленный порядковый номер, *mayack* – это булевская переменная, указывающая, разрешено ли в данном состоянии послать подтверждение для номера *ackseqnum*. Полная структура данных для получателя выглядит следующим образом.

Receiver:

1) $output : finite\_sequence[Frames]$,
2) $rwindow : \{0, 1, \ldots RW - 1\} \rightarrow (seqnum : nat, frn : Frames \cup \{\varepsilon\})$,
3) $ackseqnum : nat$,
4) $mayack : bool$

Чтобы в окне получателя могли находиться как кадры, так и пустые элементы, мы используем мощный механизм, доступный в PVS – *абстрактные типы данных*. Переменная frn, используемая в определении получателя, принадлежит к типу PVS $FrameNothing$. Этот абстрактный тип данных (полное определение которого здесь не даётся) включает значения двух видов – $frame$ (то есть определённый кадр) и $nothing$ (то есть пустой элемент). Заметим, что $frame$ и $nothing$ являются *конструкторами*, полностью перечисляющими элементы типа $FrameNothing$, а $frame?$ и $nothing?$ – соответствующими этим элементам *распознавателями*. Доступ к параметрам конструкторов осуществляется посредством *деструкторов*. В

данном случае у конструктора $frame$ имеется единственный деструктор $frid$, получающий значение кадра. Из этих определений следует, что для произвольного кадра fr выполняется: $frame?(frame(fr)) = true, nothing?(frame(fr)) = false$ и $frid(frame(fr)) = fr$.

Оба канала моделируются очередями с неограниченной вместимостью, которые могут терять сообщения. Формально канал кадров представляется конечной последовательностью сообщений с кадрами, а канал подтверждений – конечной последовательностью сообщений с подтверждениями. Все сообщения содержат порядковый номер, а сообщения с кадрами – также и кадр. Определение структуры сообщений имеет следующий вид.

FrameMessage: 1) $seqnum : nat$,
2) $frame : Frames$

AckMessage: 1) $ackseqnum : nat$

Полное состояние протокола $State$ состоит из структур данных для отправителя, получателя и двух каналов $frameChannel$ и $ackChannel$. Начальное состояние протокола определяется так, что большинству переменных даётся значение 0 или пустая последовательность.

State:

1) $sender : Sender$,
2) $receiver : Receiver$,
3) $frameChannel : finite\_sequence[FrameMessage]$,
4) $ackChannel : finite\_sequence[AckMessage]$

В нашем протоколе используются 7 атомарных операций: две для отправителя ($send$ и $receiveAck$), три для получателя ($receive$, $deliver$ и $sendAck$), одна для канала кадров ($loseFrame$) и одна для канала подтверждений ($loseAck$). Здесь мы даём лишь неформальное описание параметров, предусловия и эффекта этих операций.

- $send$ – это операция как для начальной, так и повторной отправки кадров. Она содержит в качестве параметра порядковый номер посылаемого кадра во входной последовательности. Для данной структуры данных нет необходимости включать сам кадр как параметр, потому что он легко может быть определён из своего порядкового номера во входной последовательности.

- $receiveAck$ – это операция для получения подтверждений отправителем. Она имеет три параметра: сообщение с подтверждением $ackmes$, булевская переменная $accept$, указывающая, принимается ли данное сообщение или отклоняется, и порядковый номер кадра во входной последовательности, которому соответствует данное подтверждение. Параметр $accept$ здесь необходим потому, что только принятые сообщения меняют состояние отправителя, в то время как отклонённые сообщения просто удаляются из канала. Такое удаление сообщений может быть реализовано и путём применения операции $loseAck$, однако мы полагаем, что немедленное избавление от нежелательных сообщений более точно соответствует неформальному определению протокола.

- $receive$ – это операция для получения кадров получателем. Она имеет три параметра: сообщение с кадром $frammes$, булевская переменная $accept$, указывающая, принимается ли данное сообщение или отклоняется, и наконец,

позиция в окне получателя (то есть некоторый номер от 0 до RW – 1), в которую должен быть размещён кадр из данного сообщения в случае его принятия. Так же как и для операции *receiveAck*, параметр accept здесь необходим, так как только принятые сообщения влияют на состояние получателя.

- Операция *deliver* доставляет принимающему узлу первый по порядку кадр из окна получателя.
- *sendAck* – это операция, посылающая подтверждение для последнего кадра, доставленного принимающему узлу.
- Операция $loseFrame$ удаляет сообщение с определённым порядковым номером из канала кадров.
- Операция *loseAck* удаляет сообщение с определённым порядковым номером из канала подтверждений.

Чтобы лучше продемонстрировать наш метод спецификации, здесь мы показываем предусловие и эффект для операции *deliver*, не имеющей параметров. Эффект данной операции будет использован при доказательстве свойства безопасности в следующем разделе. Результат даётся в императивном стиле, близком к спецификациям PVS.

При определении эффекта *deliver* используется дополнительная функция $addLastFrame$: если fr – это кадр и $frFS$ – это конечная последовательность кадров, то $addLastFrame(fr, frFS)$ – это конечная последовательность кадров, в которой fr добавлена к концу $frFS$. Ещё одна функция $shift$ определяет сдвиг окна получателя, т.е. она удаляет доставляемый кадр из начала окна, а в его конец добавляет пустой элемент с порядковым номером, на единицу превышающим непосредственно предшествующий ему номер. В данном определении предполагается, что в настоящий момент протокол находится в некотором состоянии $s0$, из которого он совершает переход в новое состояние $s1$.

Предусловие *deliver*:
$frame?(frn(rwindow(s0)(0)))$
Эффект *deliver*:
$output := addLastFrame(frid(frn(rwindow(s0)(0))), output(s0))$,
$rwindow := shift(rwindow(s0),$
$(seqnum := seqnum(rwindow(s0)(RW-1)) + 1, frn := nothing))$,
$ackseqnum := seqnum(rwindow(s0)(0))$,
$mayack := TRUE$

Как видно из этих определений, операция *deliver* может выполняться лишь в том случае, если в самом начале окна получателя находится кадр, а не пустой элемент. Если данное условие выполняется, то операция *deliver* добавляет данный начальный кадр к выходной последовательности. Естественно, при этом также должно быть сдвинуто окно получателя, т.е. из него убирается доставляемый кадр и освобождается место для размещения нового кадра. Также порядковый номер доставляемого кадра записывается в переменную $ackseqnum$, чтобы можно было начать рассылку подтверждений для него.

4. Спецификация и верификация свойства безопасности

Протокол скользящего окна корректен по отношению к безопасности, если получатель всегда доставляет кадры принимающему узлу в том же порядке, в каком они находятся во входной последовательности. В нашей модели мы предпочитаем определять корректность в терминах состояний, а не операций. Заметим, что в каждом состоянии кадры, уже доставленные принимающему узлу, представляются выходной последовательностью. Поэтому свойство безопасности для конкретного состояния s может быть выражено предикатом, который означает, что выходная последовательность является префиксом входной последовательности (в PVS элементы последовательности нумеруются, начиная с 0):

$Safe(s) = \forall i : i < length(output(s)) \Rightarrow output(s)(i) = input(s)(i)$

Как уже было определено ранее, $st(r)$ и $act(r)$ обозначают соответственно последовательность состояний и последовательность операций в исполнении r. Мы определяем пробег r как безопасный, если он безопасен в каждом своём состоянии (где j – номер элемента последовательности):

$Safety(r) = \forall j : Safe(st(r)(j))$

Чтобы установить свойство безопасности для нашего протокола, мы должны доказать в PVS следующую теорему, названную Main:

$\forall r : Safety(r)$ Main

Доказательство теоремы Main состоит примерно из 22 лемм и теорем PVS. Проверка доказательства занимает около 30 секунд на персональном компьютере.

Доказательство теоремы Main. Подобно всем доказательствам в PVS, наше доказательство имеет форму дерева. Корнем дерева является сама теорема Main, а большинство из его листьев – это леммы $IniLem$, $PreLem$ и $EffLem$, которые будут даны ниже. Эти леммы, которые мы называем элементарными леммами, следуют непосредственно из определения пробега, данного в предыдущем разделе. Заметим, что в этом определении мы должны заменить начальное состояние Ini и предикаты Pre и Eff на их значения для данного конкретного протокола $SWinit$, $SWPre$ и $SWEffect$.

Элементарная лемма $IniLem$ выражает, что первое состояние в любом пробеге должно быть равно начальному состоянию. Легко видеть, что она непосредственно следует из свойства 1 в определении исполнений.

$\forall r : st(r)(0) = SWinit$ IniLem

Лемма PreLem означает, что каждая операция с индексом i должна быть разрешена предикатом предусловия в состоянии с тем же индексом. Она следует из свойства 2 в определении исполнений.

$\forall r, i : SWPre(st(r)(i), act(r)(i))$ PreLem

Наконец, элементарная лемма EffLem выражает, что каждая операция с индексом i должна преобразовывать состояние с тем же индексом в соответствии с предикатом эффекта. Она следует из свойства 3 в определении исполнений.

$\forall r, i : SWEffect(st(r)(i), act(r)(i), st(r)(i+1))$ Efflem

Продолжим доказательство теоремы. Пусть r обозначает произвольное исполнение. Обозначим как $output_r(i)$ и $input_r(i)$ соответственно выходную и входную последовательности в состоянии с индексом i, а длину выходной последовательности как $LO_r(i)$. Доказательство проводится индукцией по длине выходной последовательности. Мы доказали индукцией по k следующую теорему MainInduct, выражающую отношение между длиной выходной последовательности и свойством безопасности:

$\forall r, i, k : LO_r(i) = k \Rightarrow Safe(st(r)(i))$ MainInduct

Очевидно, что из MainInduct следует наша основная теорема. Базис индукции очевиден, потому что последовательность с длиной 0 не имеет ни одного элемента. Осталось доказать шаг индукции. Предположим, что теорема была доказана для любой длины выходной последовательности, не превышающей k и что мы находимся в таком состоянии с индексом i, что $LO_r(i) = k + 1$. Чтобы доказать $Safe(st(r)(i))$, нам понадобятся 4 дополнительные леммы L1, L2, L3 и L4, доказательства которых здесь не рассматриваются. Лемма L1 выражает, что выходная последовательность может быть изменена только операцией $deliver$:

$\forall r, i : output_r(i+1) \neq output_r(i) \Rightarrow deliver?(act(r)(i))$ L1

Лемма L2 выражает, что если длина выходной последовательности равна некоторому натуральному числу $n + 1$, то в данном пробеге имеется предшествующее состояние, в котором длина выходной последовательности увеличилась с n до $n+1$, причём выходная последовательность не изменилась в промежутке от полученного состояния до текущего состояния:

$\forall r, k, n : LO_r(k) = n + 1 \Rightarrow$
$\exists\, l : l < k \& LO_r(l) = n \& LO_r(l+1) = n + 1 \& output_r(l+1) = output_r(k)$ L2

Применяя леммы L1 и L2, мы получаем, что существует индекс l такой, что $l < i$, $LO_r(l) = k$, $LO_r(l+1) = k+1$, $act(r)(l) = deliver$ и $output_r(l+1) = output_r(i)$. Таким образом, в состоянии с индексом l мы увеличили длину выходной последовательности с k до нынешнего значения $k + 1$, и после этого увеличения выходная последовательность не изменилась. Теперь мы можем применить предположение индукции к состоянию с индексом l, и это даёт нам $Safe(st(r)(l))$. Мы разделили доказательство $Safe(st(r)(i))$ на две части (*) и (**):
(*) $Safe(st(r)(l)) \Rightarrow Safe(st(r)(l+1))$
(**)$Safe(st(r)(l+1)) \Rightarrow Safe(st(r)(i))$

Сначала мы докажем часть (**). Её доказательство основано на следующей лемме L3, выражающей, что входная последовательность остаётся неизменной во всех состояниях любого пробега:

$\forall r, i, j : input_r(i) = input_r(j)$ L3

Применяя лемму L3, мы получаем $input_r(l+1) = input_r(i)$. Но мы уже доказали, что $output_r(l+1) = output_r(i)$. Таким образом, входная и выходная последовательности остались такими же в состоянии с индексом i, как и в состоянии с индексом $l+1$. Значит, если выходная последовательность была префиксом входной в состоянии с индексом $l+1$, то это соотношение между ними сохранилось и в состоянии с индексом i. Это завершает доказательство части (**).

В оставшейся части этого доказательства мы обозначаем как $Input_r$ входную последовательность в произвольном состоянии исполнения r, а как $Input_r(i)$ её элемент с индексом i.

Теперь мы докажем часть (*). Её доказательство использует дополнительную лемму L4. Введём следующие обозначения: для исполнения r и состояния в нём с индексом i, пусть $SNum_r(i,m)$ обозначает порядковый номер, находящийся на позиции m в окне получателя (где $m < RW$), а $Frn_r(i,m)$ обозначает соответствующий этому порядковому номеру кадр или пустой элемент.
$SNum_r(i,m) = seqnum(rwindow(st(r)(i))(m))$
$Frn_r(i,m) = frn(rwindow(st(r)(i))(m))$

Используя эти обозначения, мы определим предикат $SafeRWindow_r$. Его неформальный смысл таков: он выполняется в состоянии с индексом i, если для любого натурального числа m (где $m < RW$), порядковый номер, находящийся на позиции m в окне получателя, равен сумме m и нынешней длины выходной последовательности. При этом если данному порядковому номеру соответствует кадр, а не пустой элемент, то он является элементом *входной* последовательности с аналогичным индексом, равным сумме m и текущего значения LO_r.
$SafeRWindow_r(i) = \forall m : m < RW \Rightarrow SNum_r(i,m) = LO_r(i) + m\&$
$(frame?(Frn_r(i,m)) \Rightarrow frid(Frn_r(i,m)) = Input_r(LO_r(i) + m))$

Если предикат $SafeRWindow_r$ выполняется в состоянии с индексом i, то мы говорим, что это состояние *имеет безопасное окно получателя.*

Лемма L4 выражает тот факт, что любое состояние любого пробега имеет безопасное окно получателя (параметр k используется в формулировке леммы для того, чтобы облегчить её доказательство по индукции):

$$\forall r, i, k : LOr(i) = k \Rightarrow SafeRWindow_r(i) \qquad \text{L4}$$

Рассмотрим доказательство части (*) теоремы MainInduct. Мы уже знаем, что в состоянии с индексом l была выполнена операция $deliver$. Применяя элементарную лемму PreLem для этой операции, мы получаем, что в состоянии $st(r)(l)$, на позиции 0 в окне получателя находился кадр, а не пустой элемент. Применяя теперь для этой же операции элементарную лемму EffLem, а также определение сокращения Frn_r, мы получаем: $output_r(l+1) = addLastFrame(frid(Frn_r(l,0)), output_r(l))$. Теперь мы можем использовать лемму L4 при $i = l$ и $k = LO_r(l)$. Отсюда следует, что состояние с индексом l имеет безопасное окно получателя. Если теперь в предикате $SafeRWindow_r$ мы подставим $m = 0$, это даёт нам $frid(Frn_r(l,0)) = Input_r(LO_r(l))$. Но мы уже знаем, что $LO_r(l) = k$, и это приводит к равенству $output_r(l+1) = addLastFrame(Input_r(k), output_r(l))$. Теперь определение функции $addLastFrame$ в совокупности с тем, что $Safe(st(r)(l)) = true$ и $LO_r(l) = k$, приводит к заключению, что в состоянии с индексом $l+1$ выходная последовательность

по-прежнему является префиксом входной. Это завершает доказательство части (*) и всей теоремы MainInduct.

5. Заключение

Верификация протокола скользящего окна уже рассматривалась в значительном числе публикаций, и поэтому здесь мы можем перечислить лишь некоторые из них. В одной из первых работ [9] о протоколе SWP содержится лишь неформальное доказательство корректности. Формализованная верификация протокола была дана в [3] с использованием алгебры процессов, а доказательство корректности было автоматизировано с помощью PVS. Однако данное доказательство нельзя считать полным, так как оно использует несколько сложных результатов об алгебрах процессов, которые не были доказаны в PVS.

Интересная верификация протокола скользящего окна была также представлена в [8]. Использованный там подход к спецификации протокола, основанный на расширенных автоматах, имеет определённое сходство с методом, изложенным в данной статье, причём доказательство свойства корректности также было автоматизировано с помощью системы PVS. Однако стоит отметить, что в [8] кадры моделируются натуральными числами, в то время как мы представляем их произвольным типом данных. Это различие делает наше моделирование значительно более общим. По сравнению с [8], в нашей спецификации используются более естественные структуры данных. Например, в нашей спецификации в основном встречаются конечные последовательности, в то время как в [8] используются лишь бесконечные последовательности. Заметим, что использование конечных последовательностей, легко программируемых с помощью массивов, делает нашу спецификацию ближе к возможным реализациям протокола.

Для упрощения доказательства, в отличие от [2], мы предполагаем в данной работе, что в изучаемом протоколе нет ограничений на размер сообщений. Поэтому в рассматриваемой версии при отправке каждого кадра отправитель использует его первоначальный индекс во входной последовательности, а не его остаток по какому-то модулю. Заметим, что аналогичное упрощение протокола используется в работе [8], а также в некоторых других работах, посвящённых верификации SWP.

Список литературы

1. **Алексеев И.В., Соколов В.А., Чалый Д.Ю.** Моделирование и анализ транспортных протоколов в информационных сетях. Ярославль: Ярославский государственный университет, 2004. 262 с.

2. **Таненбаум Э. Компьютерные сети**. СПб.: Питер, 2002. 848 с.

3. **Badban B., Fokkink W., Groote J.F., Pang J., van de Pol J.** Verification of a Sliding Window Protocol in μCRL and PVS // Formal Aspects of Computing. 2005. V. 17(3). P. 342–388.

4. **Chkliaev D.A.** Mechanical verification of concurrency control and recovery protocols (PhD thesis) // Eindhoven: Eindhoven University of Technology, 2001. 154 p. Available at http://alexandria.tue.nl/extra2/200112908.pdf.

5. **Chkliaev D., Hooman J., de Vink E.P.** Verification and Improvement of the Sliding Window Protocol // Lecture Notes in Computer Science. 2003. V. 2619. P. 113–127.

6. **Chkliaev D.A., Nepomniaschy V.A.** Specification and verification of the classical sliding window protocol // Bulletin of the Novosibirsk Computing Center, Series: Computer Science, IIS Special Issue. 2011. V. 32. P. 37–56.

7. **Owre S., Rushby J.M., Shankar N.** PVS: a Prototype Verification System // Lecture Notes in Computer Science. 1992. V. 607. P. 748–752.

8. **Rusu V.** Verifying a Sliding-Window Protocol using PVS. // Formal Description Techniques (FORTE'01). 2001. P. 251–266.

9. **Stenning N.V.** A data transfer protocol // Computer Networks. 1976. V. 1. P. 99–110.

Deductive Verification of the Sliding Window Protocol

Chkliaev D.A., Nepomniaschy V.A.

Keywords: communication protocols, Sliding Window Protocol, fault tolerance, formal specification, automated verification, interactive theorem proving, PVS

We consider the well-known Sliding Window Protocol which provides reliable and efficient transmission of data over unreliable channels. A formal proof of correctness for this protocol faces substantial difficulties caused by a high degree of parallelism which creates a significant potential for errors. Here we consider a version of the protocol that is based on selective repeat of frames. The specification of the protocol by a state machine and its safety property are represented in the language of the verification system PVS. Using the PVS system, we give an interactive proof of this property of the Sliding Window Protocol.

Сведения об авторах:
Шкляев Дмитрий Александрович
Институт систем информатики имени А.П. Ершова СО РАН,
магистр математики, ведущий программист;
Непомнящий Валерий Александрович,
Институт систем информатики имени А.П. Ершова СО РАН,
кандидат физ.-мат. наук, зав. лабораторией.