И. Бурдонов
Докторская диссертация. Доклад.
ИСПРАН, доклады на семинаре 20 февраля, 18 марта и 21 мая на защите. 2008

ДОКЛАД

1 слайд.

Теория конформности для функционального тестирования программных систем на основе формальных моделей.

2 слайд.

Цель работы

Создание теории конформности, которая:

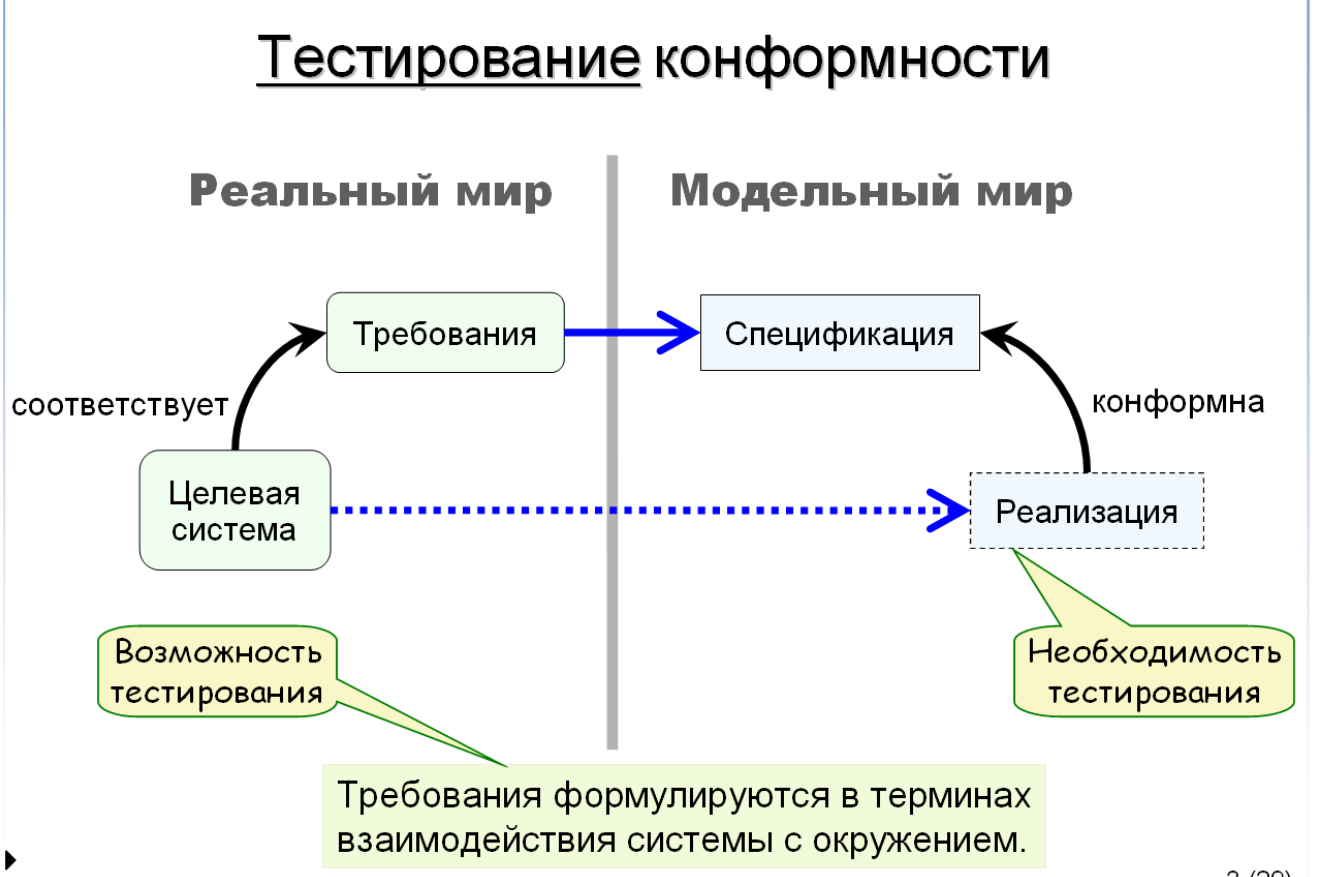
1. Применима для класса семантик тестового взаимодействия, основанных на наблюдаемом поведении при функциональном тестировании, с параметризацией конформности («сводимости» реализации к спецификации) семантикой из этого класса.
2. Учитывала все типы поведения, ненаблюдаемого или запрещенного при тестировании, при условии, что такое поведение не препятствует проверке конформности заданной спецификации в тестовых экспериментах.
3. Развита до уровня алгоритмов генерации тестов по спецификации.
4. Решает проблемы сохранения конформности при асинхронном тестировании (тестировании в контексте) и автоматической верификации декомпозиции системных требований для сложных систем, состоящих из многих компонентов.

Игорь Борисович Бурдонов, ИСП РАН

Целью работы было создание теории конформности, которая отвечала бы четырём требованиям, показанным на этом слайде. Более подробно об этих требованиях, связанных с ними проблемах и способах их решения речь пойдёт дальше.

3 слайд.

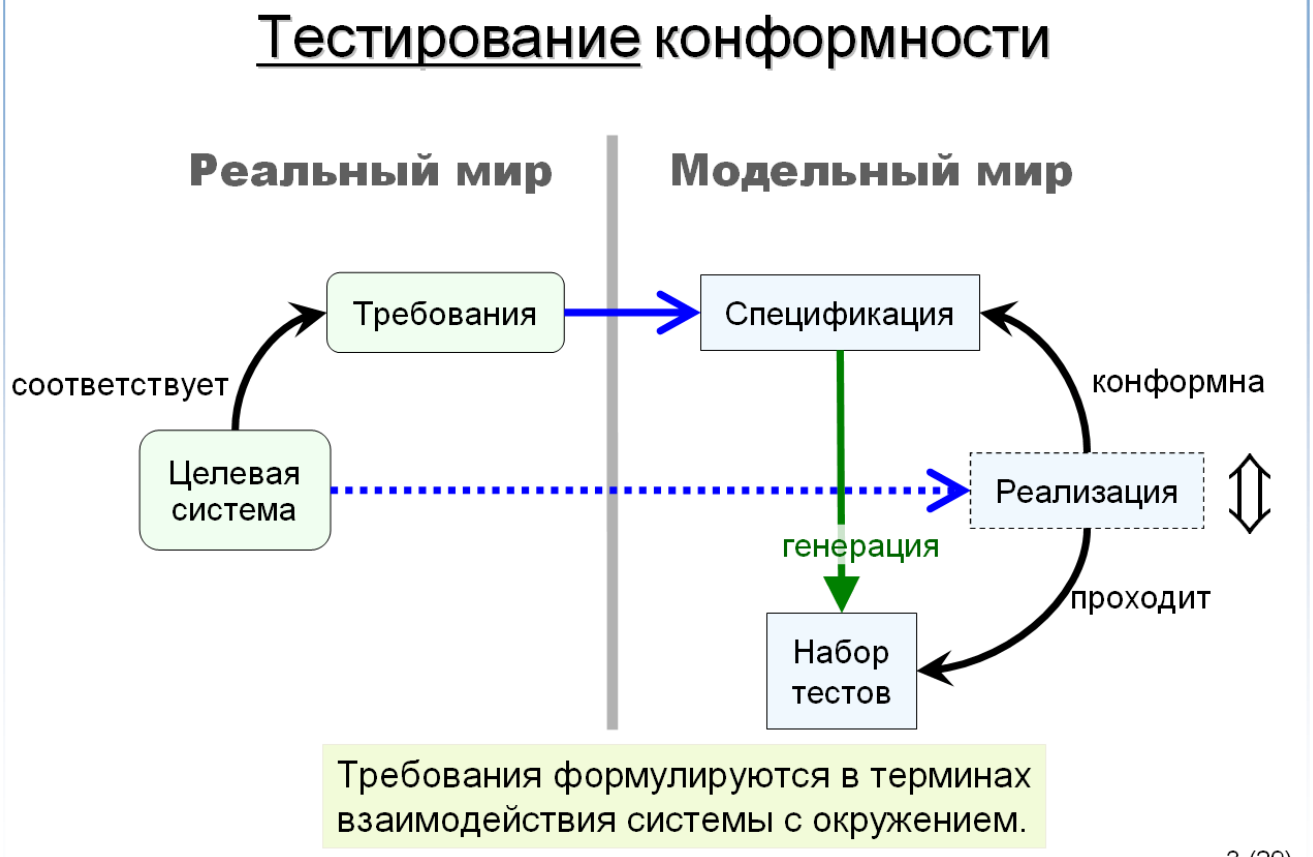
Под верификацией понимается проверка правильности исследуемой системы как её соответствия заданным требованиям. На формальном уровне такое соответствие означает, что модель системы и модель требований связаны заданным бинарным отношением. Модель системы я буду называть реализационной моделью или (для краткости) *реализацией*, модель требований – *спецификацией*, а их соответствие – отношением *конформности*.



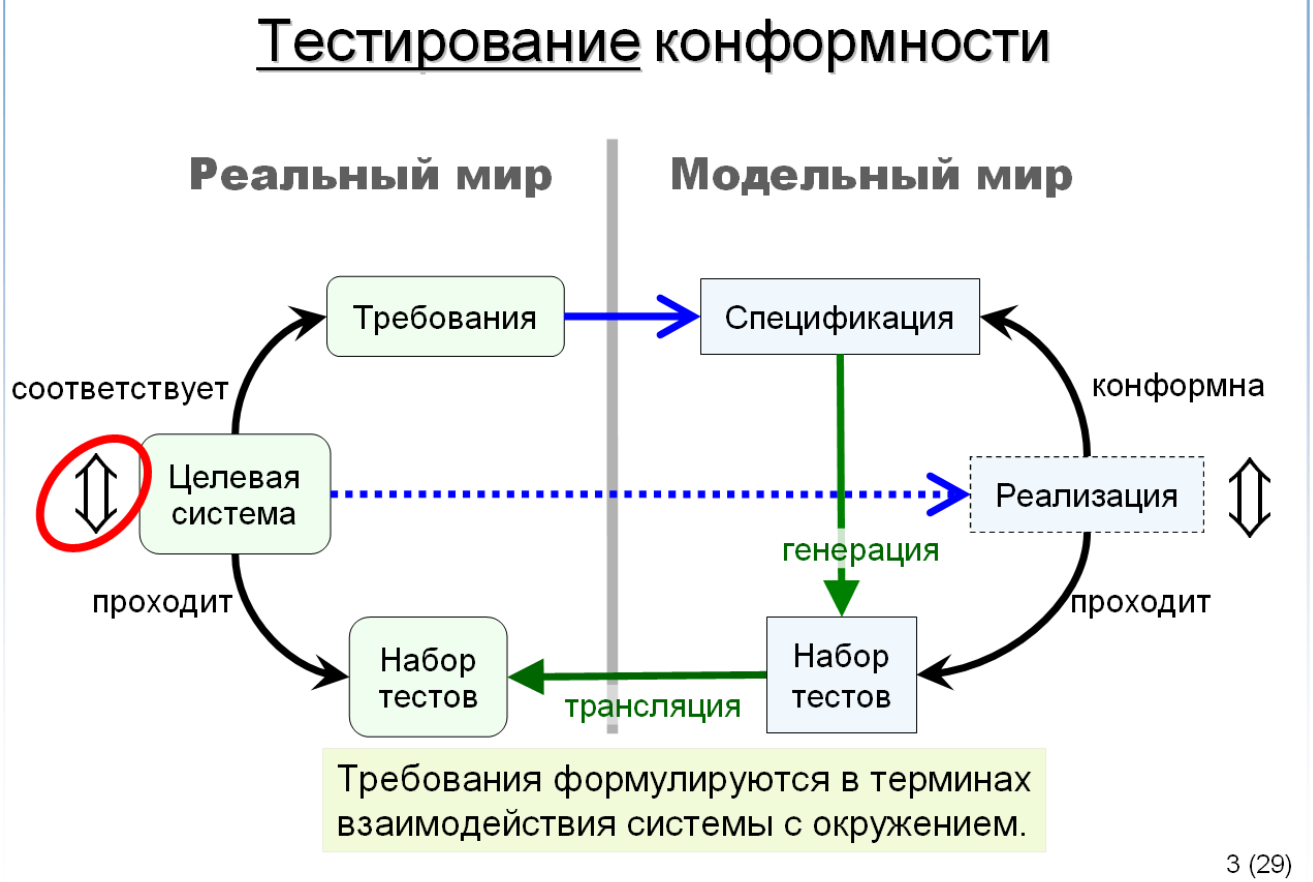
Тестирование – это верификация с помощью эксперимента.

Когда тестирование **необходимо**? Когда реализационная модель не задана или слишком сложна для анализа, и методы аналитической верификации не применимы. Тем не менее, предполагается, что реализационная модель существует – это, так называемая, тестовая гипотеза.

Когда тестирование **возможно**? Когда требования к системе выражены в терминах её взаимодействия с внешним миром (окружением).



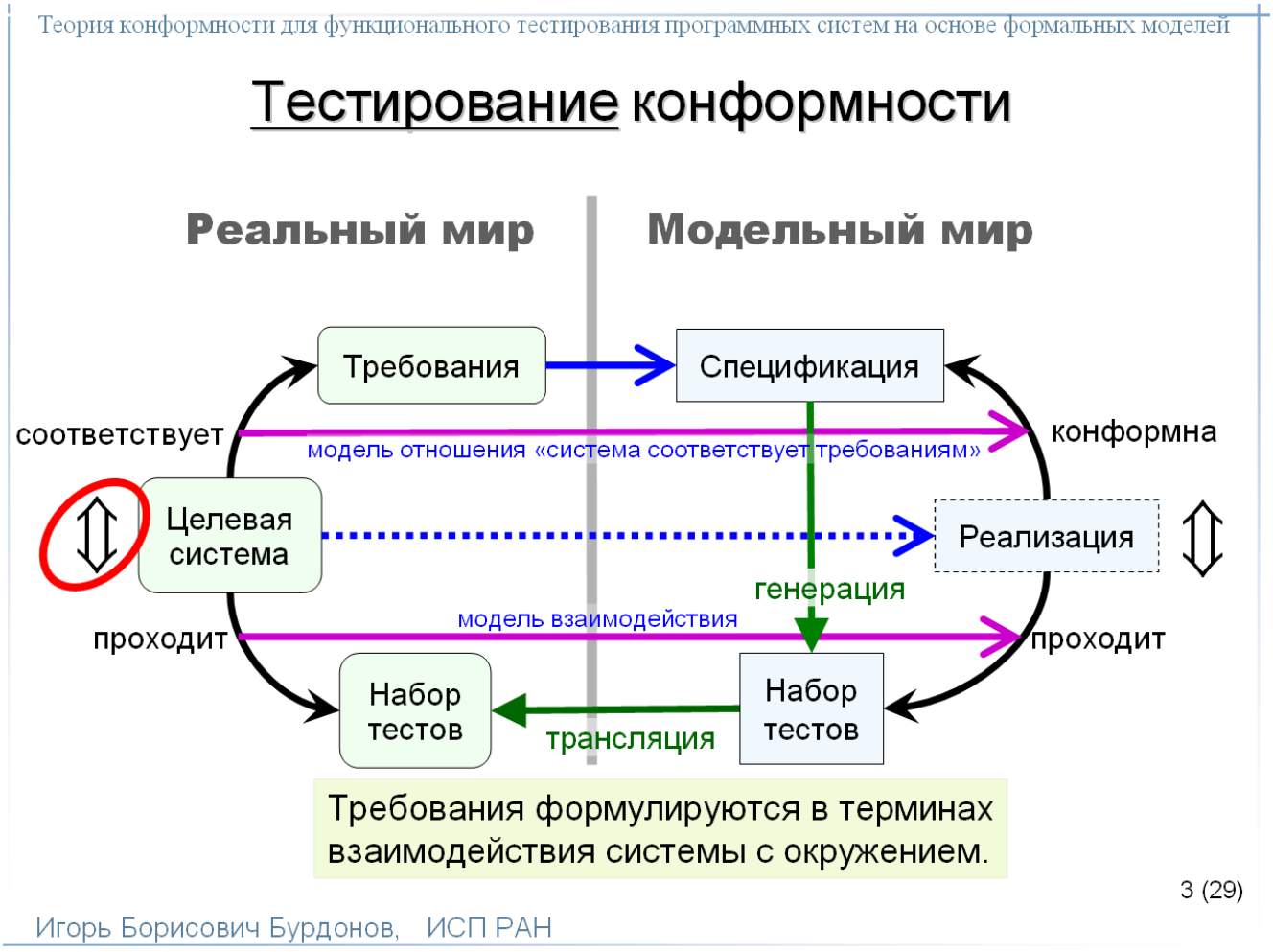
Основной задачей является генерация по заданной спецификации модельных тестов. Тесты должны для любой реализационной модели проверить, конформна она заданной спецификации или нет. Эта проверка основана на модели взаимодействия системы с её окружением. Вводится модельное отношение «реализация проходит тест». Набор тестов принято называть полным, если реализация проходит каждый тест из набора тогда и только тогда, когда она конформна спецификации. Как написано в последней книге Владимира Васильевича Липаева, такие тесты можно считать вторым эталоном, или второй формой описания функциональности системы. Второй – после спецификации.



После того, как тесты созданы, они транслируются в реальные тестовые программы, которые, взаимодействуя с целевой системой, устанавливают её соответствие или несоответствие исходным требованиям.

Эта схема работы основана на предположениях трёх типов:

1) Правильно моделируются объекты – синие стрелки.



2) Правильно моделируются отношения – фиолетовые стрелки.
3) Правильно делаются преобразования – зелёные стрелки.

Вот этими стрелками сейчас и займёмся.

4 слайд.

→ Модели

усложнение ↓

Модель	Проблемы тестирования
Функция	Большое число значений аргумента, в т.ч. большие данные со сложной структурой.
Автомат	+ Перебор (ненаблюдаемых) состояний.
LTS	+ Сложное взаимодействие (недетерминизм, отказы, дивергенция).

Сети Петри, алгебраические и контрактные спецификации.

Причины выбора LTS: 1) распространённость,
2) удобство генерации тестов,
3) определена композиция моделей.

Игорь Борисович Бурдонов, ИСП РАН

Сначала о моделях – синие стрелки.

На этом слайде изображены три типа моделей и соответствующие им проблемы тестирования по мере усложнения.

Самая простая модель – математическая функция как абстракция чистой процедуры. Основная проблема здесь – большое количество значений аргумента, которые нужно проверить для того, чтобы убедиться, что результат всегда правильный. Особенно, когда аргумент – это большие данные со сложной структурой как, например, для компиляторов.

Вторая модель – это автомат, взаимодействие с которым сводится к той же самой простой схеме «стимул-реакция», что и для функции. Здесь добавляется проблема перебора

состояний. Сложность в том, что сами состояния не наблюдаемы при функциональном тестировании.

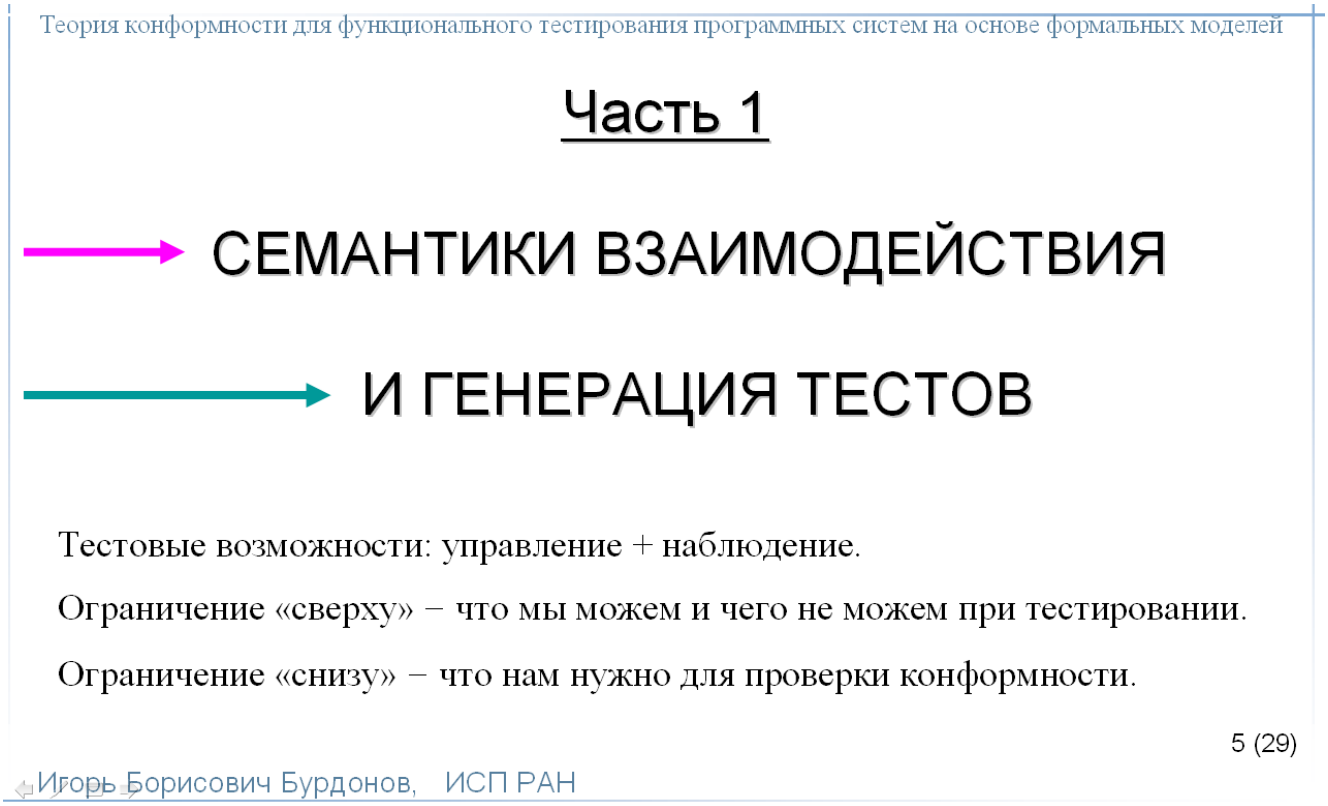
Третья модель – это система помеченных переходов, по-английски, Labelled Transition System, LTS. Для таких систем характерно сложное взаимодействие. Теперь в ответ на стимул может быть несколько реакций или ни одной. Можно подавать несколько стимулов подряд или получать реакции без стимулов. Для LTS характерны такие явления как недетерминизм, отказы, когда стимулы не принимаются или реакции не выдаются, а также дивергенция – зацикливание, когда система выполняет какую-то внутреннюю, ненаблюдаемую работу и не взаимодействует с тестом. В работе внимание концентрируется именно на этом.

Для систем последнего типа применяются и другие модели: сети Петри, алгебраические спецификации, в частности ASM – Abstract State Machine, а также контрактные спецификации, построенные на пред- и постусловиях.

Можно назвать три причины, по которым в данной работе выбрана модель LTS:

- Распространённость этой модели.
- Удобство генерации тестов по LTS.
- И то, что на LTS определена композиция, моделирующая взаимодействие компонентов составной системы.

5 слайд.

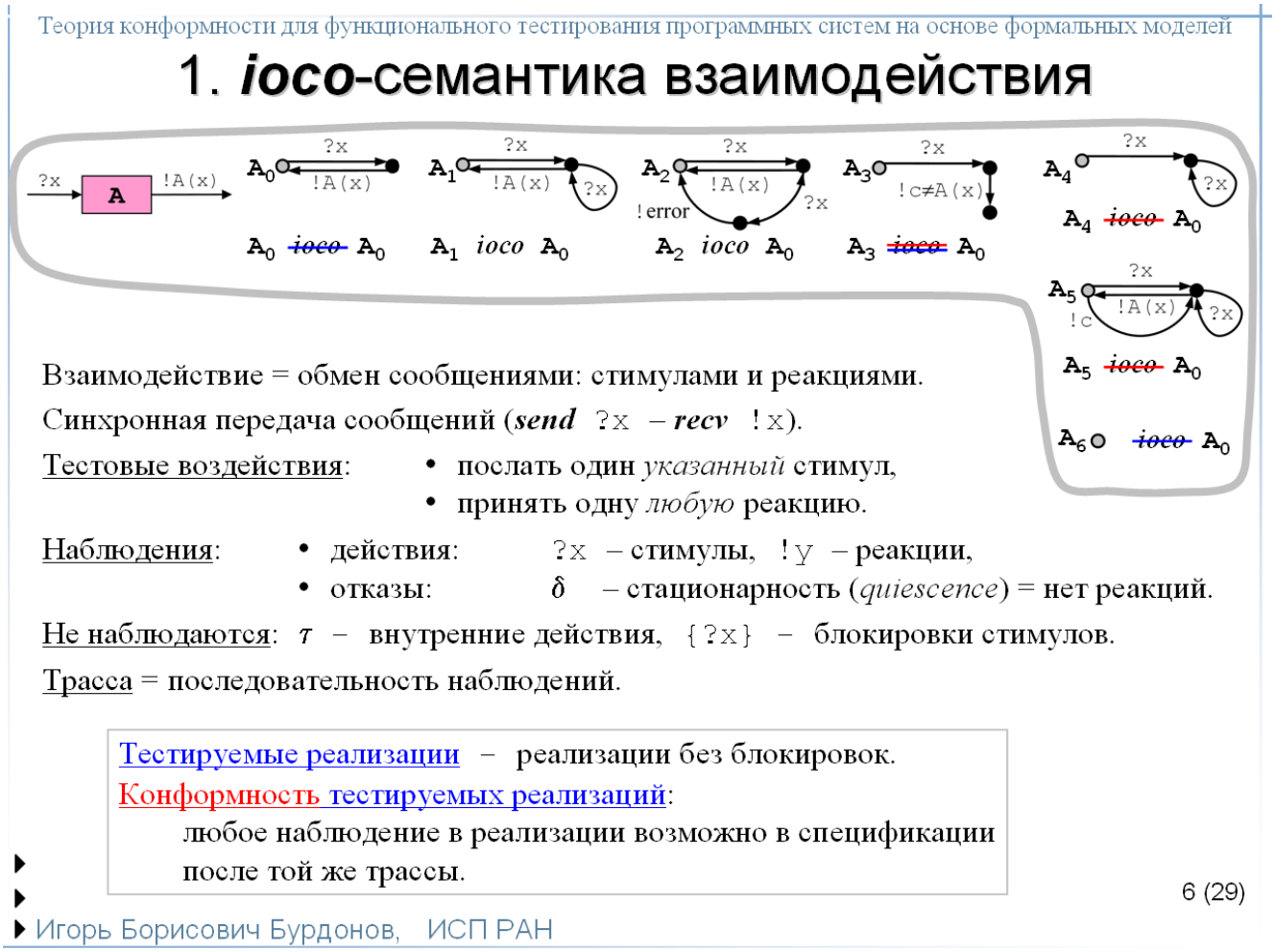


Теперь речь пойдёт о фиолетовых стрелках, то есть о том, как моделируются соответствие требованиям и тестовое взаимодействие. Потом – о генерации тестов.

Семантика тестового взаимодействия строится на пересечении того, что мы можем, и того, что нам нужно. Речь идёт о тестовых возможностях: как мы можем воздействовать на реализацию и что можем наблюдать. То, что мы можем или не можем, – это ограничение тестовых возможностей «сверху». То, что нам требуется для проверки конформности, – это ограничение «снизу».

Я приведу ряд примеров, подобранных так, чтобы, во-первых, продемонстрировать эти ограничения «сверху» и «снизу».
Во-вторых, примеры максимально просты, чтобы довести идею в чистом виде.
В-третьих, каждый пример является представителем целого класса часто встречающихся случаев, важных для практики.

6 слайд.



Начнём с семантики отношения конформности ***ioco***. Оно расшифровывается как *Input-Output Conformance*. Предложено Яном Тритмансом.

Взаимодействие понимается как обмен сообщениями: стимулами и реакциями в синхронном режиме, то есть без какой либо буферизации.

Тестовые воздействия. Можно либо послать в реализацию один выбранный стимул, либо принять одну, но любую из выдаваемых реализацией, реакцию.

Мы можем наблюдать либо действие (приём стимула или выдачу реакции), либо отсутствие реакций. Отсутствие реакций обозначается символом δ, называется стационарностью (по-английски, *quiescence*, молчание) и является разновидностью отказа как отсутствия действий.

Что мы не можем наблюдать? Во-первых внутренние, ненаблюдаемые действия. Они обозначаются символом τ. Во-вторых, блокировку стимула, когда реализация не принимает стимул. Это ограничение «сверху» на наши тестовые возможности.

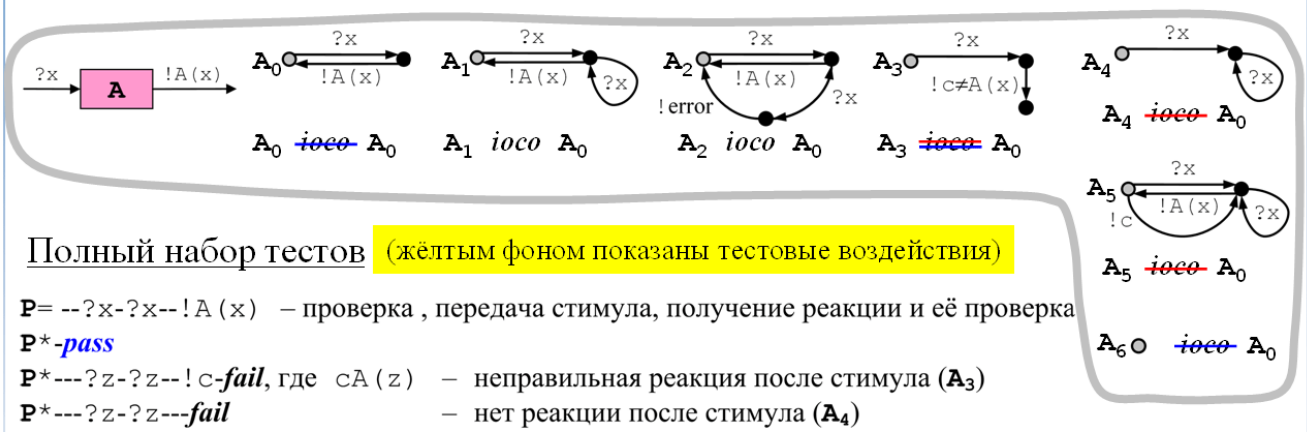
Результат тестового эксперимента – это трасса = последовательность наблюдений.

Отношение ***ioco*** является одним из отношений редукции – сводимости реализации к спецификации. Редукция означает, что любое наблюдение, которое может быть получено при тестировании реализации, возможно и в спецификации в той же самой ситуации, то есть после наблюдения той же самой трассы. Для ***ioco*** такими наблюдениями могут быть стимулы и реакции, а также отсутствие реакций – стационарность.

Принимая реакции, мы всегда что-нибудь наблюдаем: либо реакцию, либо стационарность. Посылая стимул, мы должны быть уверены, что он будет принят, так как в противном случае мы будем ждать наблюдения, а его может не быть. Отношение ***ioco*** решает эту проблему просто: предполагается, что в тестируемых реализациях нет блокировок. Реализация принимает каждый стимул в каждом достижимом состоянии, то есть она всюду определена по стимулам (*input-enabled*).

На слайде приведён пример системы, которая получает стимул x и выдаёт в качестве реакции результат вычисления функции $A(x)$. A_0 – это спецификация, кроме того показаны ещё шесть реализаций с A_1 по A_6. Синим зачёркиванием отмечены реализации, которые не входят в домен отношения ***ioco***, поскольку в них есть блокировки, а красным – неконформные реализации.

1. ***ioco*-семантика взаимодействия**



Полный набор тестов (жёлтым фоном показаны тестовые воздействия)

P= --?x-?x--!A(x) – проверка , передача стимула, получение реакции и её проверка

P*-***pass**

P*---?z-?z--!c-***fail**, где cA(z) – неправильная реакция после стимула (**A**$_3$)

P*---?z-?z---***fail** – нет реакции после стимула (**A**$_4$)

P*--!c-***fail** – реакция без стимула (**A**$_5$)

Некоторые тестовые инструменты для ***ioco***:

TGV (Test Generation with Verification technology) интегрирован в CADP (Construction and Analysis of Distributed Processes Software Tools for Designing Reliable Protocols and Systems),

TestGen (входной язык LOTOS, генерирует тест для верификации hardware design и компонентов),

TorX (генерация тестов, выполнение их и анализ результатов тестирования on-fly).

Тестируемые реализации – реализации без блокировок.

Конформность тестируемых реализаций:
любое наблюдение в реализации возможно в спецификации после той же трассы.

6 (29)

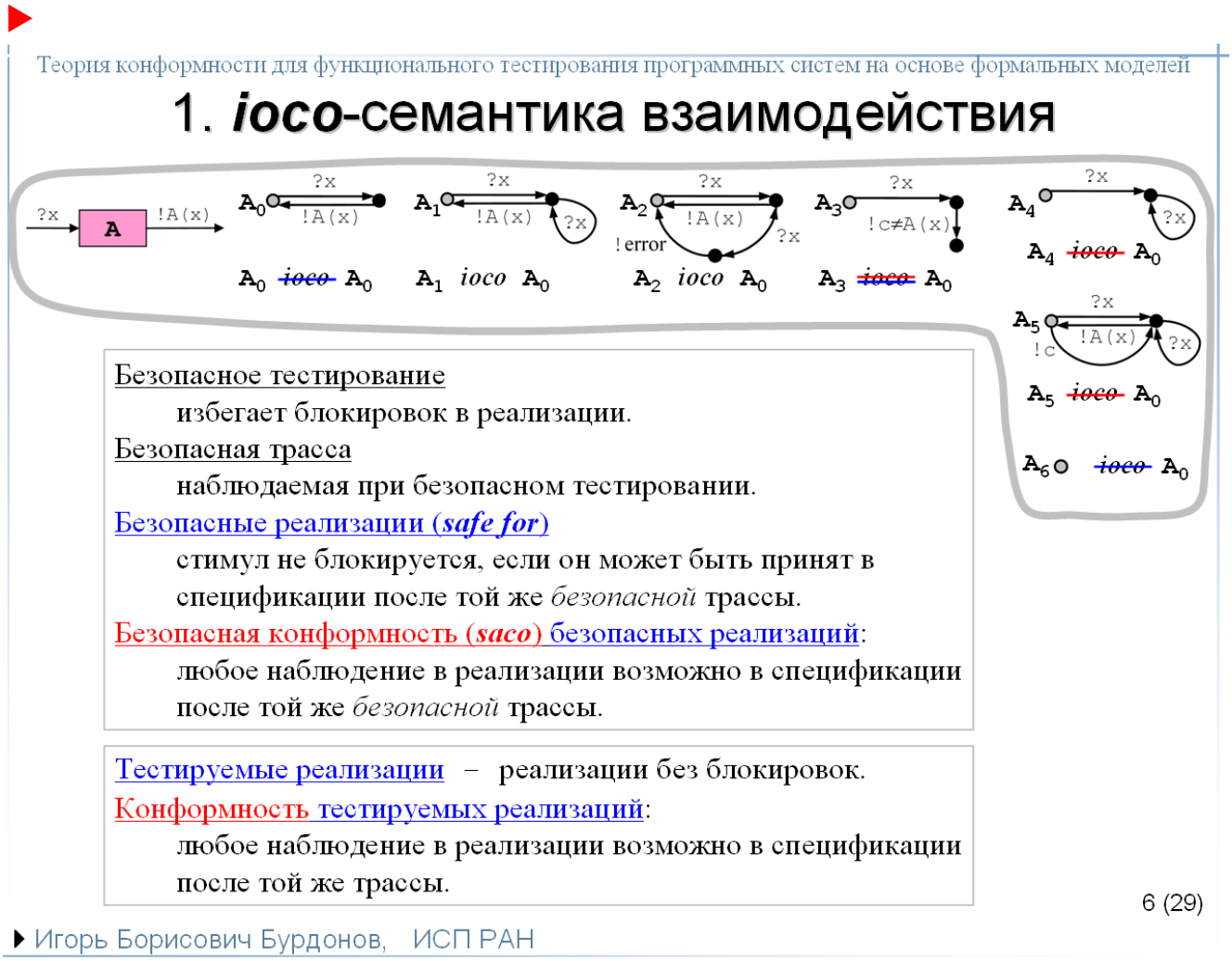
Игорь Борисович Бурдонов, ИСП РАН

Здесь приведён полный набор тестов для спецификации A_0 и отношения ***ioco***. Наборы тестов, генерируемые существующими инструментами, проверяют ровно то же самое, что и этот набор.

Теперь заметим, что любой тест, проверяющий конформность по ***ioco***, можно запускать и для A_0: ненаблюдаемой блокировки не возникнет. Более того, такой тест не обнаружит никаких ошибок и вообще не отличит реализации A_0, A_1 и A_2. Причина в том, что мы не посылаем в реализацию стимул после трассы, если в спецификации эта трасса не продолжается этим стимулом: нам нечего проверять после этого стимула. Тем самым, мы не проверяем, что делает реализация: блокирует стимул или принимает его с тем или иным дальнейшим поведением.

Тем не менее, формально реализация A_0 неконформна, поскольку не входит в домен отношения ***ioco***.

Это недостаток отношения ***ioco***.



Чтобы избавиться от этого недостатка, в работе предлагается концепция безопасного тестирования, которое избегает блокировок стимулов. Трассы, которые при этом наблюдаются, названы безопасными. Реализация безопасно-тестируема (или просто безопасна), если она не блокирует стимул, который в спецификации может быть принят в той же самой ситуации, то есть после той же самой безопасной трассы. В определении конформности «трассу» заменяем на «безопасную трассу».

1. *ioco*-семантика взаимодействия

Безопасное тестирование
избегает блокировок в реализации.

Безопасная трасса
наблюдаемая при безопасном тестировании.

Безопасные реализации (***safe for***)
стимул не блокируется, если он может быть принят в спецификации после той же *безопасной* трассы.

Безопасная конформность (***saco***) безопасных реализаций:
любое наблюдение в реализации возможно в спецификации после той же *безопасной* трассы.

Тестируемые реализации – реализации без блокировок.

Конформность тестируемых реализаций:
любое наблюдение в реализации возможно в спецификации после той же трассы.

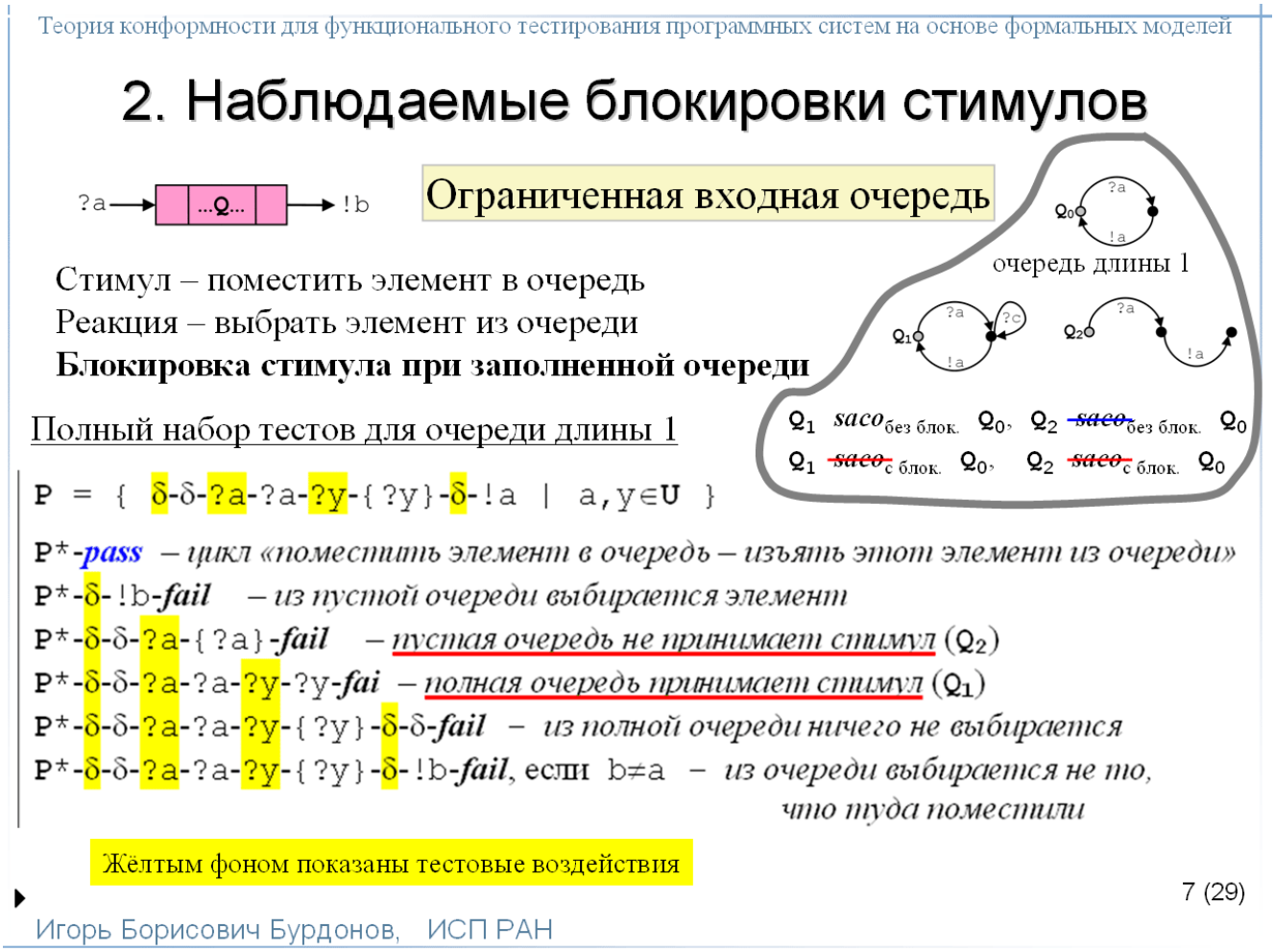
6 (29)

Игорь Борисович Бурдонов, ИСП РАН

Тогда получается вот такая картина. Теперь можно тестировать не только реализацию A_0, но и реализацию A_3, обнаруживая в ней ошибку.

Набор тестов остаётся тот же самый, поскольку различие только в расширении класса тестируемых реализаций.

7 слайд.



Теперь рассмотрим «ответвления» от ***ioco***-семантики, требующие другого отношения конформности и других методов генерации тестов. Эти ответвления соответствуют четырём часто обсуждаемым проблемам тестирования.

В первой семантике блокировки стимулов наблюдаемы. На слайде приведён пример, когда наблюдаемость блокировок требуется для проверки конформности, то есть это ограничение «снизу».

Рассмотрим очередь. Здесь стимул – это помещение элемента в очередь, а реакция – выборка элемента из очереди. Если очередь ограниченной длины, то её функциональность требует блокировки стимула, когда очередь полностью заполнена. Понятно, что это невозможно проверить с помощью тестов без наблюдаемых блокировок. Для них реализация Q_1 конформна, хотя она принимает стимул, когда очередь заполнена.

Реализация Q_2, наоборот, блокирует стимул, когда очередь становится пустой, но эту реализацию нельзя безопасно тестировать.

Если блокировки наблюдаемы, то обе реализации можно тестировать и находить в них ошибки.

Понятно, что ограниченная очередь является представителем большого класса сред передачи ограниченной ёмкости.

▶ Другие примеры:

В окне графического интерфейса высвечивается меню, в котором некоторые кнопки «бледные», то есть соответствующие стимулы блокируются.

Или автомат по продаже «чего-нибудь»: когда это «чего-нибудь» кончилось, перекрывается щель для приёма монет, что соответствует блокировке.

8 слайд.

Вторая семантика разрешает посылать сразу несколько стимулов, из которых реализация сама выбирает тот стимул, который она принимает.

В примере система F получает два аргумента на два входных порта, вычисляет функцию от них и возвращает результат на один выходной порт.

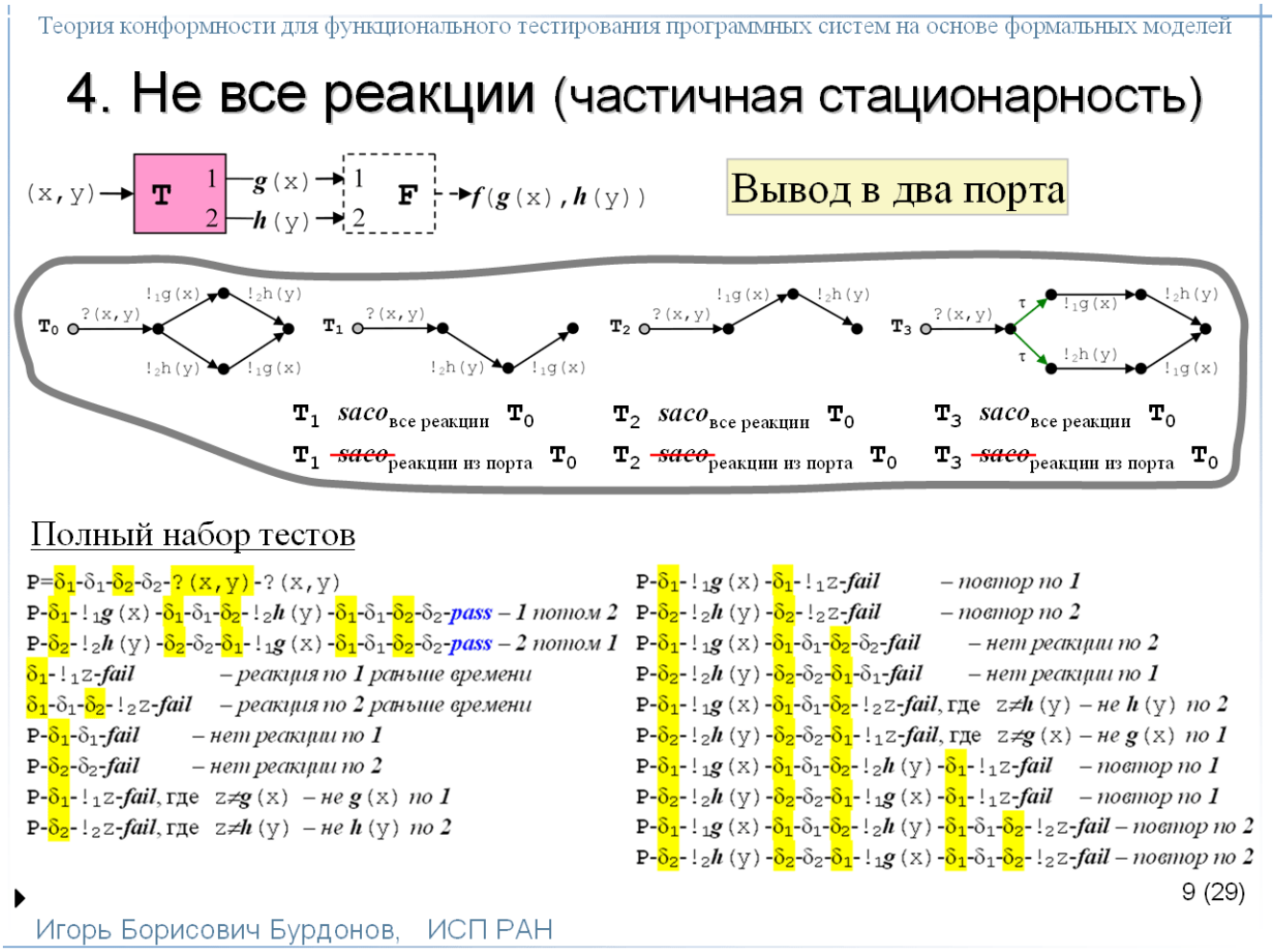
Мы хотим сформулировать требования к системе так, чтобы дать реализатору максимальную свободу. Разрешим принимать аргументы в любом порядке, лишь бы до приёма обоих аргументов реакций не было, а после был один правильный результат. Спецификация – это F_0, все четыре реализации, включая саму спецификацию, должны быть конформны. Блокировки будем считать ненаблюдаемыми: это ограничение «сверху».

Если посылать стимулы по одному, то при любом порядке их передачи в двух из четырёх реализаций будут проблемы с ненаблюдаемой блокировкой. Безопасно-тестируема только сама спецификация F_0.

Нам нужна семантика, требующая посылать сразу два аргумента: тогда блокировок не возникает, поскольку каждая реализация готова принять хотя бы один аргумент из этих двух.

Этот пример представляет класс пороговых систем, которые сначала принимают несколько стимулов на разные входные порты, а потом выдают реакцию.

9 слайд.



Третья семантика позволяет принимать не все реакции, а только часть их.

Рассмотрим систему T, которая принимает пару аргументов по одному входному порту и выдаёт значения функций от этих аргументов по двум выходным портам. Спецификация – это система T_0.

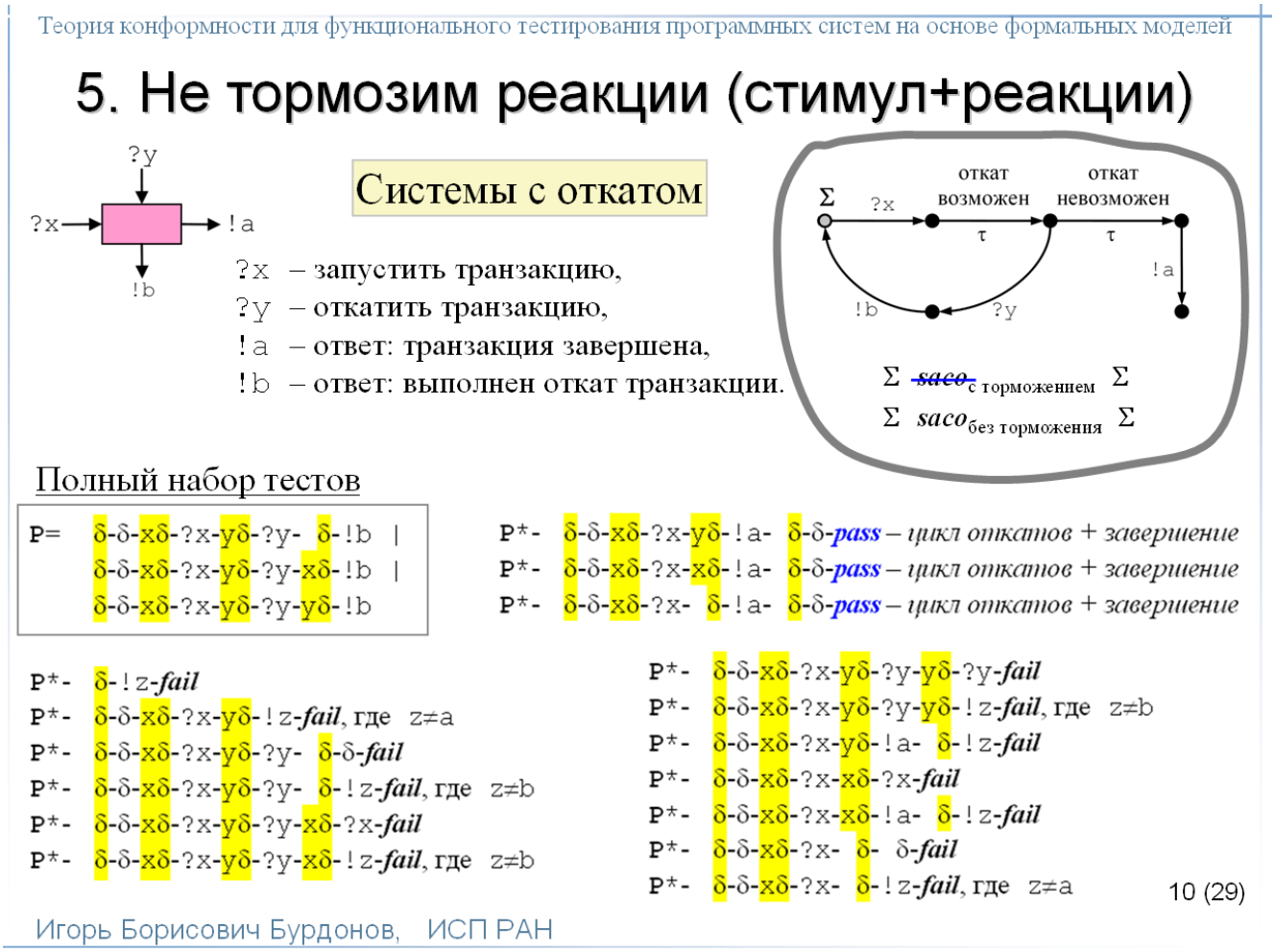
Если тест принимает все реакции, то не только сама спецификация, но и три остальных реализации на этом слайде конформны: они отличаются порядком выдачи результатов.

Однако, если мы хотим, чтобы такая система T сопрягалась с системой F, которую мы рассматривали как пример для посылки нескольких стимулов, то годится только реализация T_0, а остальные ошибочны.

Чтобы обнаруживать эти ошибки, нужно принимать реакции по отдельности из каждого порта.

- То же самое имеет место при тестировании широковещания. Например, в системе АС-6 сигнал «я есть» должен был передаваться по каждому порту, то есть каждой машине.

10 слайд.



Следующая проблема – запрет на торможение реакций. Суть её в том, что тест должен всегда принимать реакции от реализации. Это означает, что стимул посылается только с одновременным приёмом реакций. Я хочу привести пример, когда запрет на торможение реакций требуется для проверки конформности, то есть это ограничение «снизу».

Рассмотрим систему с откатом. Здесь два стимула: запустить транзакцию и откатить транзакцию. Блокировка стимулов считается ненаблюдаемой: это ограничение «сверху». Реакций тоже две: транзакция завершена и выполнен откат транзакции. Мы хотим, чтобы можно было откатить транзакцию до её завершения. В спецификации транзакция выполняется в два этапа, которые изображены двумя внутренними переходами. Откат возможен только в конце первого этапа.

Если посылать один стимул «откатить транзакцию», то может возникнуть ненаблюдаемая блокировка. Тем самым, интересующие нас реализации нельзя безопасно тестировать .

Выход в том, чтобы совмещать стимул-откат с приёмом реакций. Мы сохраним ограничение «сверху» и будем считать, что соответствующий отказ (стимул не принимается и реакции не выдаются) не наблюдаем. Но такой отказ не возникнет, поскольку либо принимается стимул-откат, либо выдаётся реакция о завершении транзакции.

11 слайд.

Сводная таблица семантик и конформностей

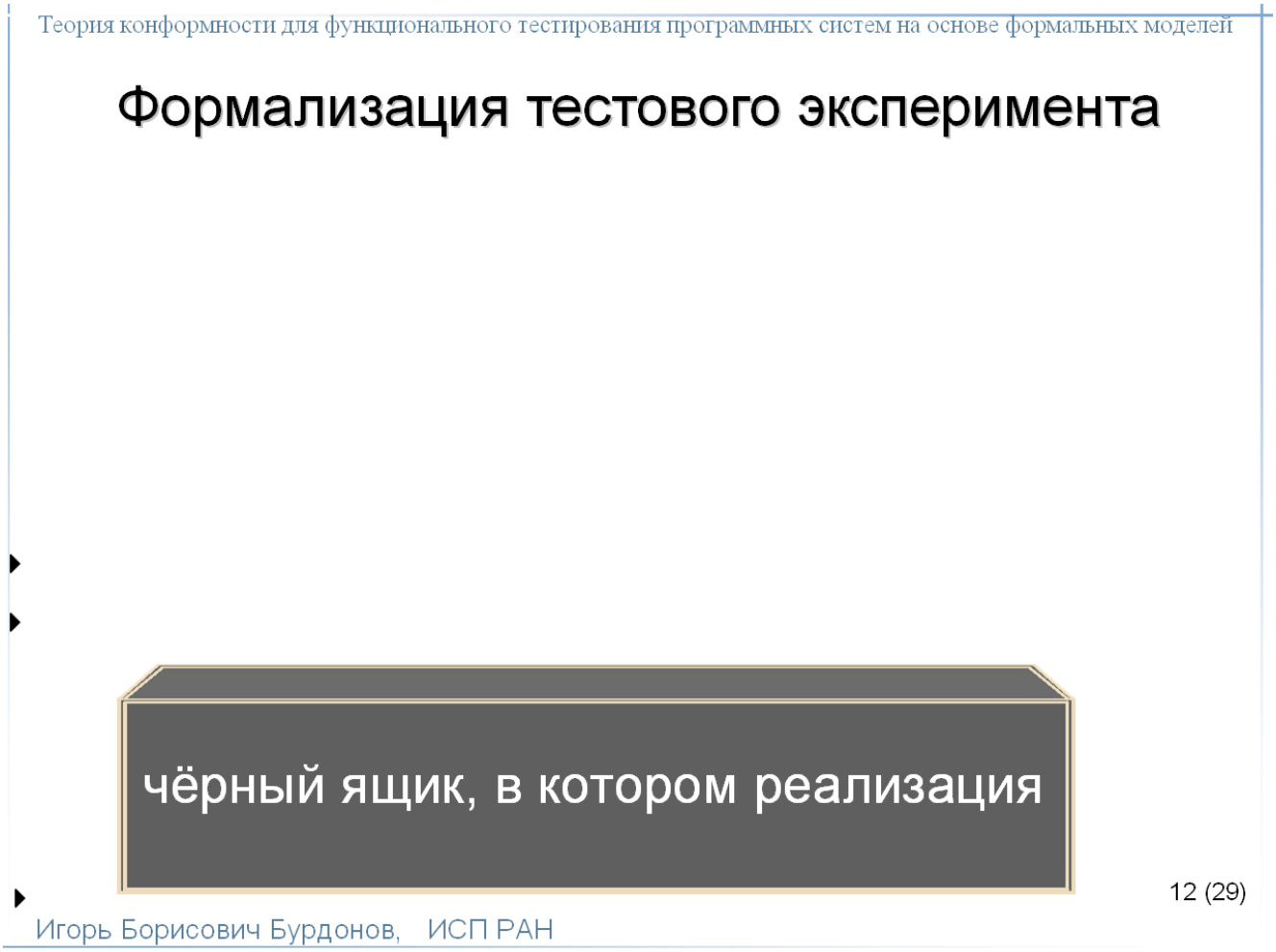
Отношения конформности	Наблюдаем блокировки стимулов	Посылаем несколько стимулов	Принимаем часть реакций	Не тормозим реакции
ioco	*нет*	*нет*	*нет*	*нет*
$ioco_{\beta\delta}$	да	*нет*	*нет*	*нет*
mioco	*почти* да	*нет*	*почти* да	*нет*
queued-quiescent trace-included (только асинхронное тестирование)	*нет*	*нет*	*нет*	да
rioco	*почти* да	*нет*	*нет*	да
–	да	*нет*	да	да
–	*нет*	*нет*	да	да/*нет*
–	да/*нет*	да	да/*нет*	да/*нет*

Все эти примеры показывают, что тестовое воздействие означает, что мы разрешаем реализации выполнить действие из указанного множества действий. Множество состоит из одного или нескольких стимулов, всех или части реакций, или смешанно: содержит как стимулы, так и реакции. Внутренние действия считаются всегда разрешёнными. В разных случаях для одних тестовых воздействий соответствующие отказы (отсутствие действий) наблюдаемы, а для других – нет.

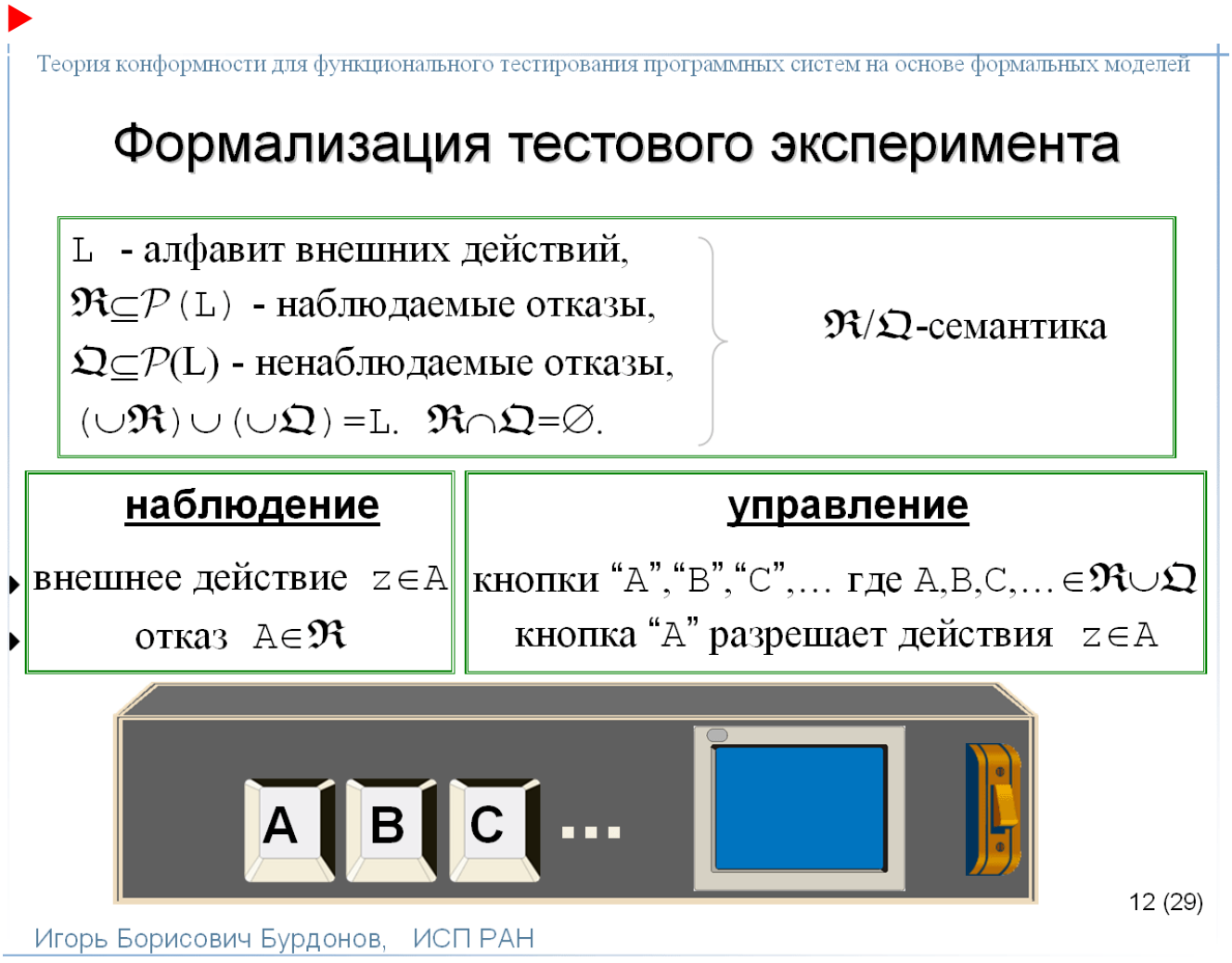
В таблице показаны сочетания рассмотренных тестовых возможностей. Для строки либо указано отношение конформности, основанное на соответствующей семантике взаимодействия и встречающееся в литературе, либо отмечено, что такая конформность пока не вводилась.

В диссертации определяется общий вид рассматриваемого класса семантик с помощью машины тестирования.

12 слайд.



Машина тестирования – это принятый способ моделирования тестового взаимодействия. Это чёрный ящик, внутри которого находится реализация.



Для управления имеется набор кнопок, а для наблюдения – дисплей.

В работе предлагается машина, в которой на кнопке написано множество разрешаемых действий. Тестовое воздействие – это нажатие кнопки. Машина параметризуется набором кнопок, то есть допустимыми тестовыми воздействиями.

В известных машинах Милнера и Ван Глаббека разрешается либо одно действие, либо любое подмножество внешних действий. Это частные случаи предлагаемой машины при соответствующем наборе кнопок.

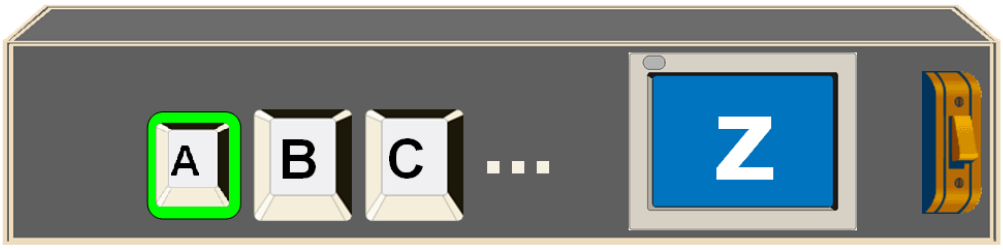
Если отказ наблюдаем, кнопка называется $\mathfrak{R}$-кнопкой, в противном случае – $\mathfrak{Q}$-кнопкой.

Формализация тестового эксперимента

L - алфавит внешних действий,
$\mathfrak{R} \subseteq \mathcal{P}(L)$ - наблюдаемые отказы,
$\mathfrak{Q} \subseteq \mathcal{P}(L)$ - ненаблюдаемые отказы,
$(\cup\mathfrak{R}) \cup (\cup\mathfrak{Q}) = L$. $\mathfrak{R} \cap \mathfrak{Q} = \varnothing$.

$\mathfrak{R}/\mathfrak{Q}$-семантика

наблюдение	**управление**
внешнее действие $z \in A$	кнопки “A”,“B”,“C”,… где $A,B,C,\ldots \in \mathfrak{R} \cup \mathfrak{Q}$
отказ $A \in \mathfrak{R}$	кнопка “A” разрешает действия $z \in A$



12 (29)

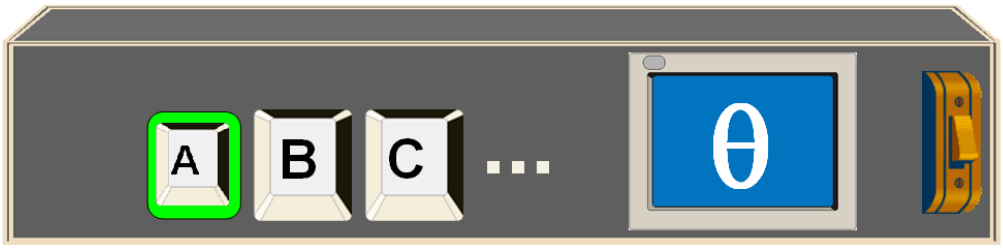
Нажимая любую кнопку с множеством **A**, мы можем наблюдать на дисплее действие **z** из этого множества.

Формализация тестового эксперимента

L - алфавит внешних действий,
$\mathfrak{R} \subseteq \mathcal{P}(L)$ - наблюдаемые отказы,
$\mathfrak{Q} \subseteq \mathcal{P}(L)$ - ненаблюдаемые отказы,
$(\cup\mathfrak{R}) \cup (\cup\mathfrak{Q}) = L$. $\mathfrak{R} \cap \mathfrak{Q} = \varnothing$.

$\mathfrak{R}/\mathfrak{Q}$-семантика

наблюдение	**управление**
внешнее действие $z \in A$	кнопки "A","B","C",... где $A,B,C,\ldots \in \mathfrak{R} \cup \mathfrak{Q}$
отказ $A \in \mathfrak{R}$	кнопка "A" разрешает действия $z \in A$



12 (29)

При нажатии $\mathfrak{R}$-кнопки может наблюдаться не только действие, но и отказ. На дисплее высвечивается специальный символ θ, а наблюдение отказа называется θ-наблюдением. Какой именно отказ мы наблюдаем, определяется нажатой кнопкой.

Задание этих наборов кнопок как раз и определяет семантику взаимодействия, которую будем называть $\mathfrak{R}/\mathfrak{Q}$-семантикой.

Результатом тестового эксперимента является трасса наблюдений: внешних действий и наблюдаемых отказов. В диссертации трассовая модель определяется как множество трасс, обладающее определённым набором свойств.

13 слайд.

На этом слайде показано соотношение машины тестирования и двух моделей.

В диссертации доказано, что они эквивалентны в следующем смысле.

1. Множество трасс наблюдений на машине тестирования является трассовой моделью.

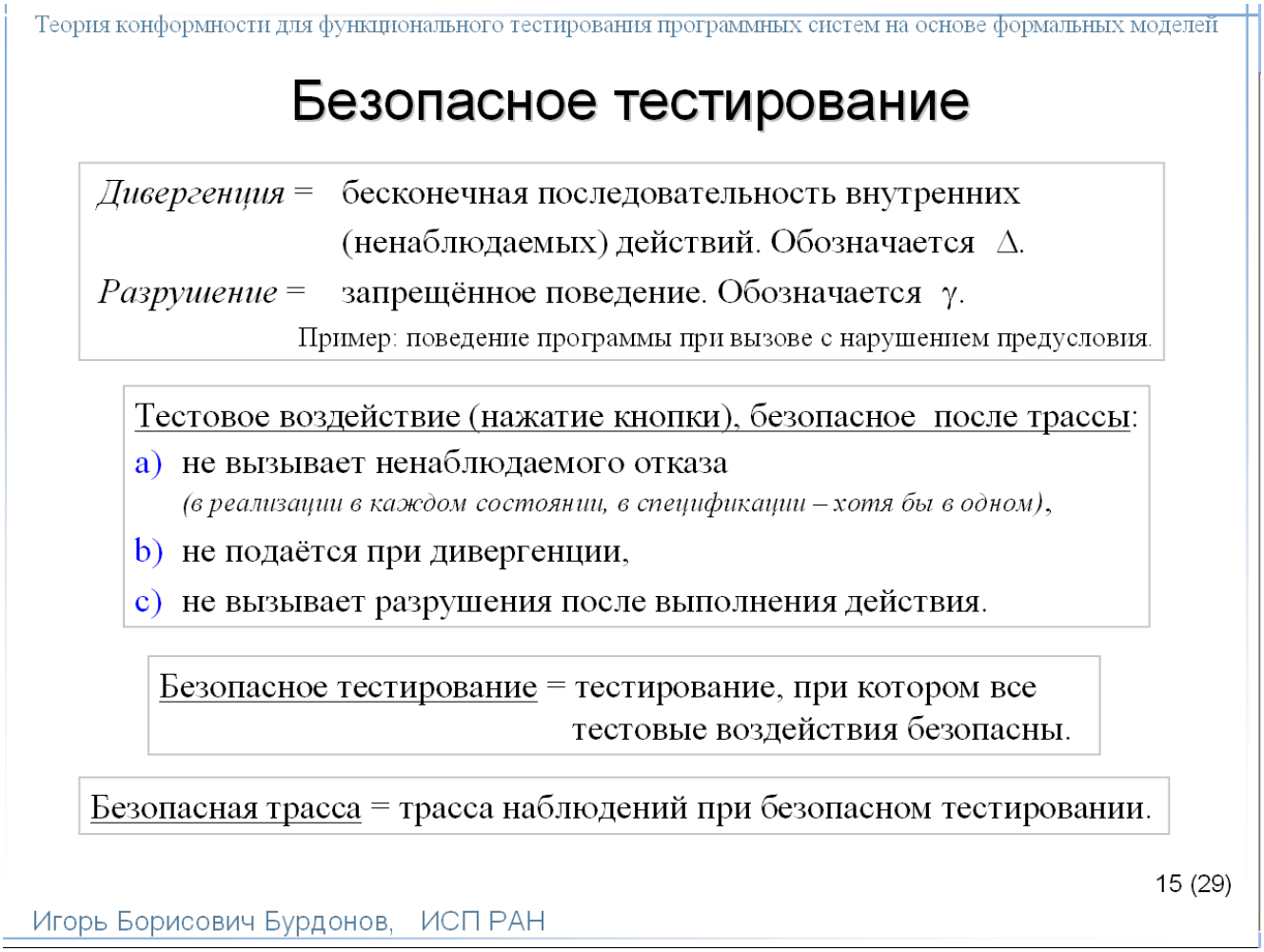
2. По трассовой модели можно построить LTS, имеющую то же множество трасс.

3. Если внутрь чёрного ящика машины тестирования поместить LTS, то будет наблюдаться множество трасс этой LTS.

14 слайд.

Здесь показано соотношение двух моделей: трассовой и LTS. Трассовой модели достаточно для определения отношений безопасности и конформности, а также для генерации тестов. LTS-модель более «подробная», она основана на понятии состояния и, тем самым, содержит избыточную, с точки зрения конформности, информацию. Зато для LTS можно определить композицию – операцию сборки составной системы из компонентов, что невозможно сделать для трасс наблюдений.

15 слайд.



До сих пор под безопасным тестированием мы понимали такое тестирование, при котором не возникают ненаблюдаемые отказы. Теперь рассмотрим ещё два понятия, относящиеся к этой теме.

Первое – это дивергенция – бесконечная последовательность внутренних действий системы, «зацикливание». При дивергенции в ответ на тестовое воздействие мы можем не получить никакого наблюдения, так как всё время будут выполняться внутренние действия. Поэтому обычно считают, что в реализации не должно быть дивергенции. В диссертации предлагается ослабить это требование: дивергенции не должно быть не вообще, а только при безопасном тестировании.

Второе понятие впервые предлагается в данной работе. Это понятие разрушения, означающее любое поведение, которое запрещено или нежелательно при тестировании. Причины

запрета могут быть самыми разными. Например, такое поведение может привести к холодному рестарту системы. Для контрактных спецификаций запрещённым можно считать поведение программы при нарушении предусловия обращения к ней.

Разрушение моделируется действием, обозначаемым символом γ.

Итак, при безопасном тестировании тестовое воздействие не подаётся, если это может вызвать ненаблюдаемый отказ, при дивергенции и в случае возможного разрушения после выполнения действия.

16 слайд.

Гипотеза о безопасности и безопасная конформность

Реализация *безопасна* для спецификации I ***safe for*** Σ :

a) Если спецификация не разрушается с самого начала, то реализация тоже не разрушается с самого начала.

b) Если после общей безопасной трассы тестовое воздействие безопасно в спецификации, то оно безопасно и в реализации.

Реализация *конформна* спецификации I ***saco*** Σ :

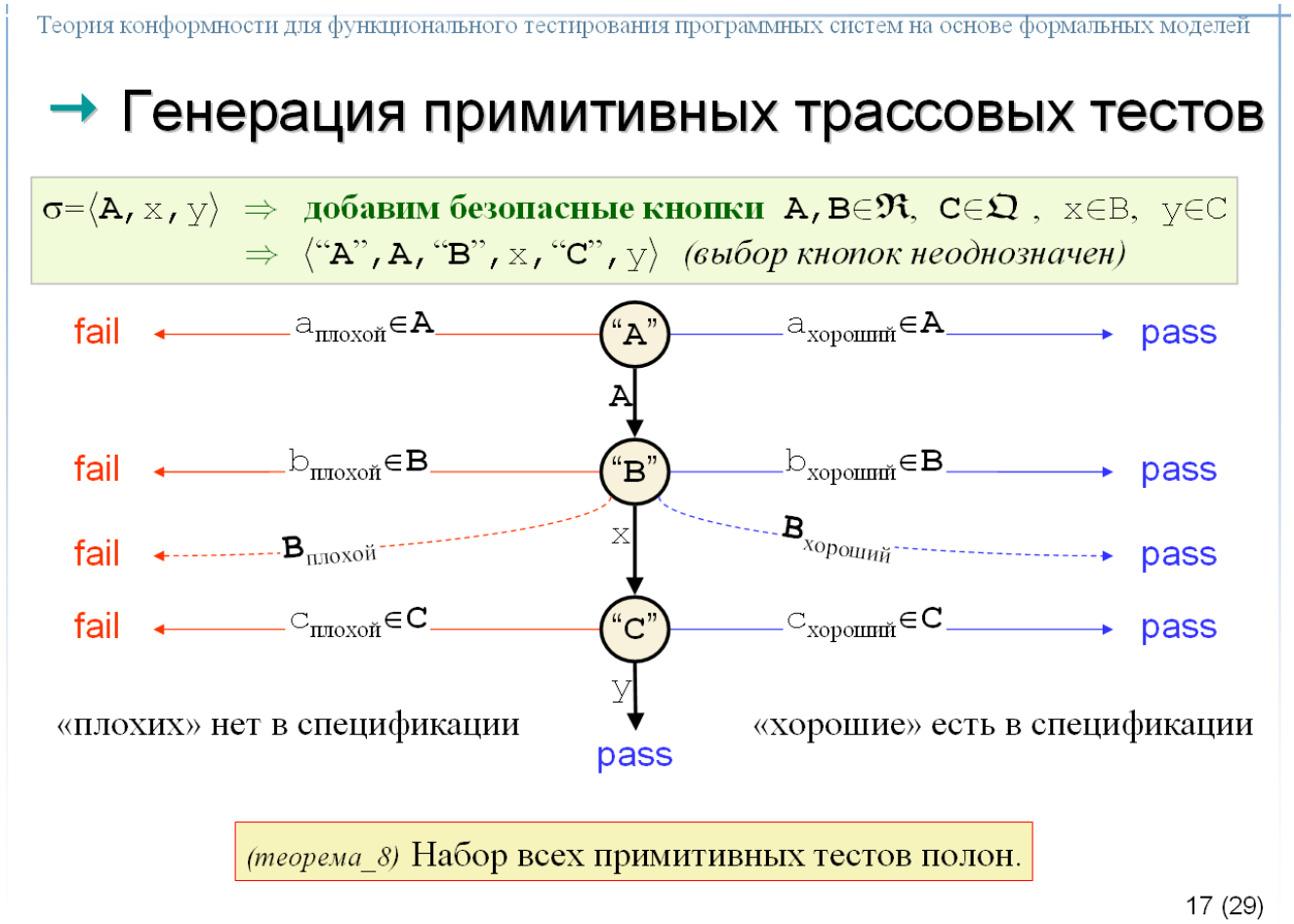
1) Предусловие тестирования: I ***safe for*** Σ.

2) Тестируемое условие: если после общей безопасной трассы выполнить тестовое воздействие, безопасное в спецификации, то любое наблюдение в реализации возможно и в спецификации.

На базе концепции безопасного тестирования определяется гипотеза о безопасности, описывающая класс безопасно-тестируемых реализаций. Суть её в том, что тестовое воздействие, которое безопасно в спецификации, должно быть безопасно и в реализации в той же самой ситуации, то есть после общей безопасной трассы.

Безопасная реализация конформна спецификации, если при безопасном тестировании любое наблюдение в реализации возможно и в спецификации после общей безопасной трассы. Это такая безопасная редукция: реализация сводится к спецификации только для безопасных трасс.

17 слайд.



Теперь про зелёные стрелки.

На слайде показан пример генерации примитивного теста в трассовой модели. Такой тест строится по одной выделенной безопасной трассе спецификации. Сначала в трассу добавляются безопасные кнопки: перед отказом A добавляется кнопка "A", а перед действием – какая-нибудь безопасная кнопка, разрешающая это действие. По получившейся чередующейся последовательности кнопок и наблюдений строится порождающий граф теста. Кнопки – нетерминальные вершины, а вердикты ***pass*** и ***fail*** – терминальные вершины. Дуга соответствует либо символу трассы, либо ответвлению. Ответвление ведёт в состояние ***pass***, если оно есть в спецификации, и в состояние ***fail*** – в противном случае.

В диссертации доказано, что набор всех примитивных тестов полный.

18 слайд.

Параллельная композиция LTS

(*à la* CCS – Calculus of Communicating Systems)

Каждому внешнему действию z поставлено в соответствие *противоположное* действие $\underline{z}$ так, что $\underline{\underline{z}}=z$.

Композиция алфавитов: $L|\!|M=(L\setminus\underline{M})\cup(M\setminus\underline{L})$.

Композиция LTS: для $\mathbf{S}=LTS(V_S, L, E_S, s_0)$ и $\mathbf{T}=LTS(V_T, M, E_T, t_0)$ $\mathbf{S}|\!|\mathbf{T}=LTS(V_S\times V_T, L|\!|M, E, s_0t_0)$, где E –порождается следующими правилами вывода:

$$z\in(L\setminus\underline{M})\cup\{\tau,\gamma\} \quad \& \quad s\text{—}z\to s` \quad \vdash \quad st\text{—}z\to s`t$$

$$z\in(M\setminus\underline{L})\cup\{\tau,\gamma\} \quad \& \quad t\text{—}z\to t` \quad \vdash \quad st\text{—}z\to st`$$

$$z\in L\cap\underline{M} \quad \& \quad s\text{—}z\to s` \quad \& \quad t\text{—}\underline{z}\to t` \quad \vdash \quad st\text{—}\tau\to s`t`$$

$$\textit{deadlock} \quad \& \quad t\text{—}\theta\to t` \quad \vdash \quad st\text{—}\tau\to st`, \text{ где}$$

$$\textit{deadlock} =_{def} s\text{—}\tau\gamma\not\to \ \& \ t\text{—}\tau\gamma\not\to \ \& \ \forall z\in L\cap\underline{M} \ (s\text{—}z\not\to \vee t\text{—}\underline{z}\not\to).$$

18 (29)

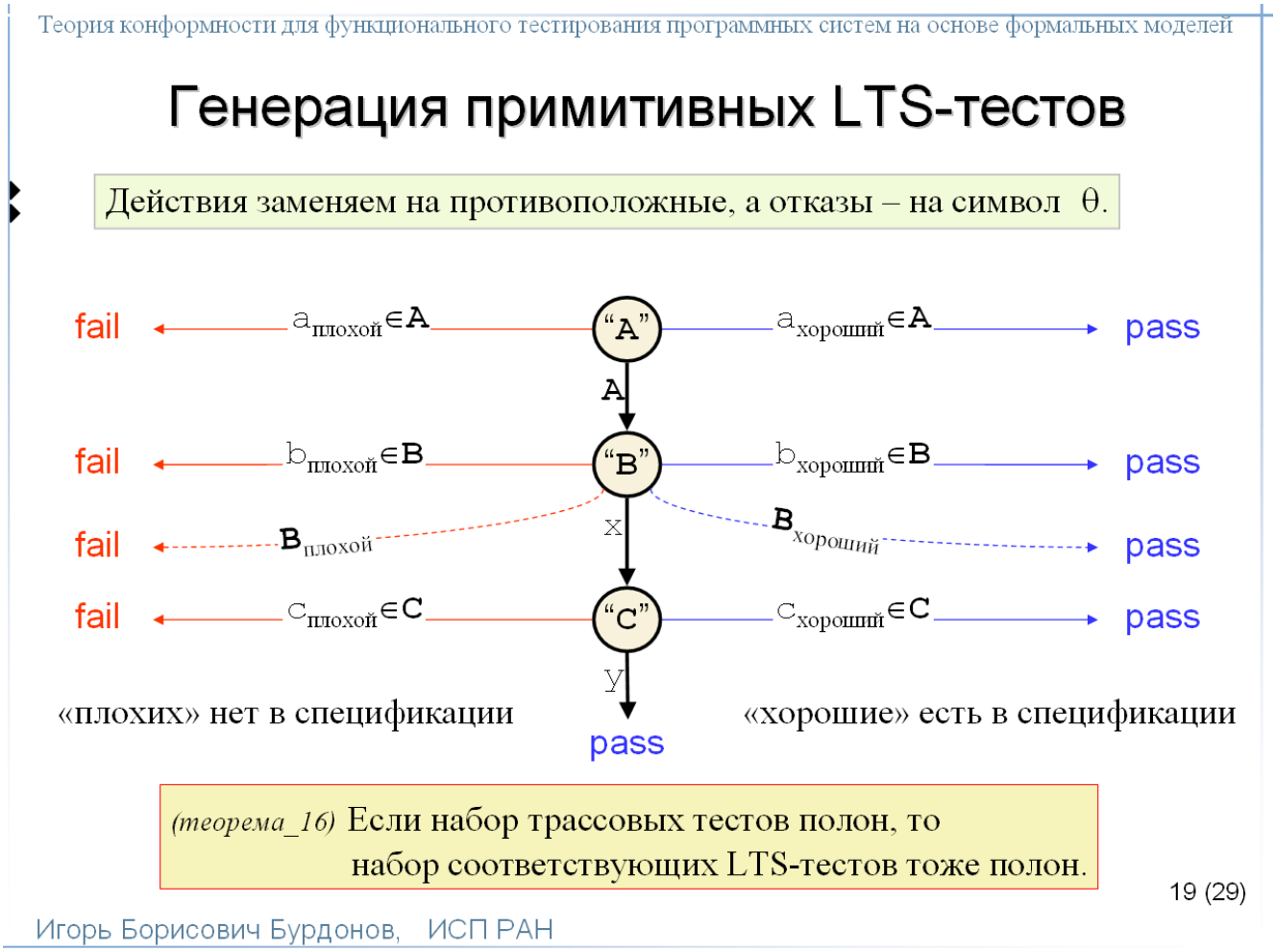
Игорь Борисович Бурдонов, ИСП РАН

Теперь перейдём к генерации LTS-тестов.

В LTS-теории взаимодействие принято моделировать оператором параллельной композиции. Суть её в том, что взаимодействие между собой двух компонентов составной системы не наблюдаемо извне системы: в композиции LTS ему соответствует внутренний τ-переход.

Кроме того, для наблюдения отказов в тесте вводятся специальные θ-переходы. Такой переход разрешает тупик во взаимодействии, когда в композиции нет других переходов, что и означает отказ как отсутствие действий.

19 слайд.

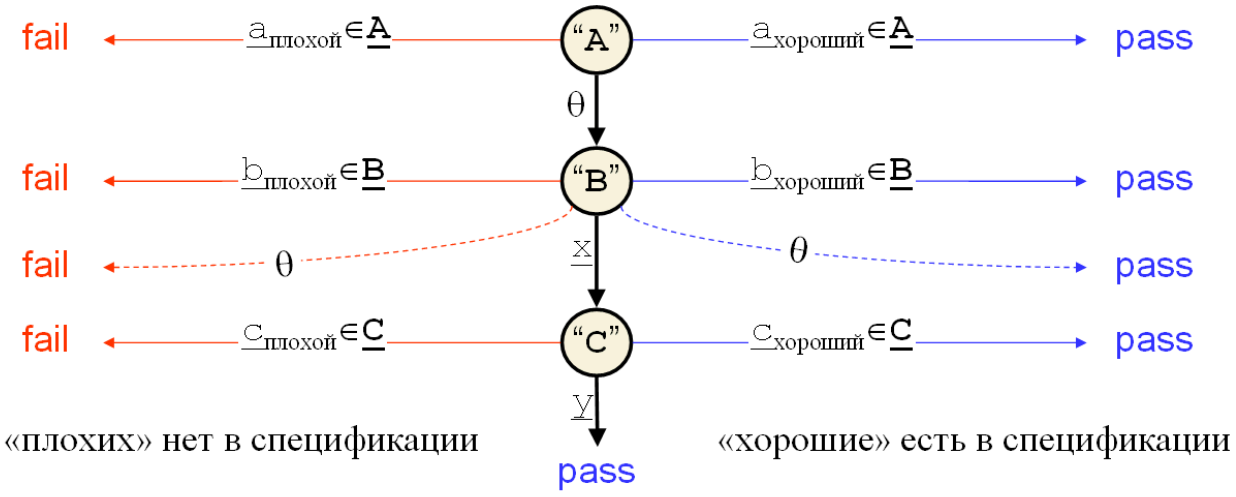


LTS-тесты в диссертации определяются через трассовые тесты. Доказывается, что, если набор трассовых тестов полный, то соответствующий набор LTS-тестов тоже полный. На слайде показан пример генерации LTS-теста по примитивному трассовому тесту.

Для этого порождающий граф трассового теста понимается как LTS.

Генерация примитивных LTS-тестов

Действия заменяем на противоположные, а отказы – на символ θ.

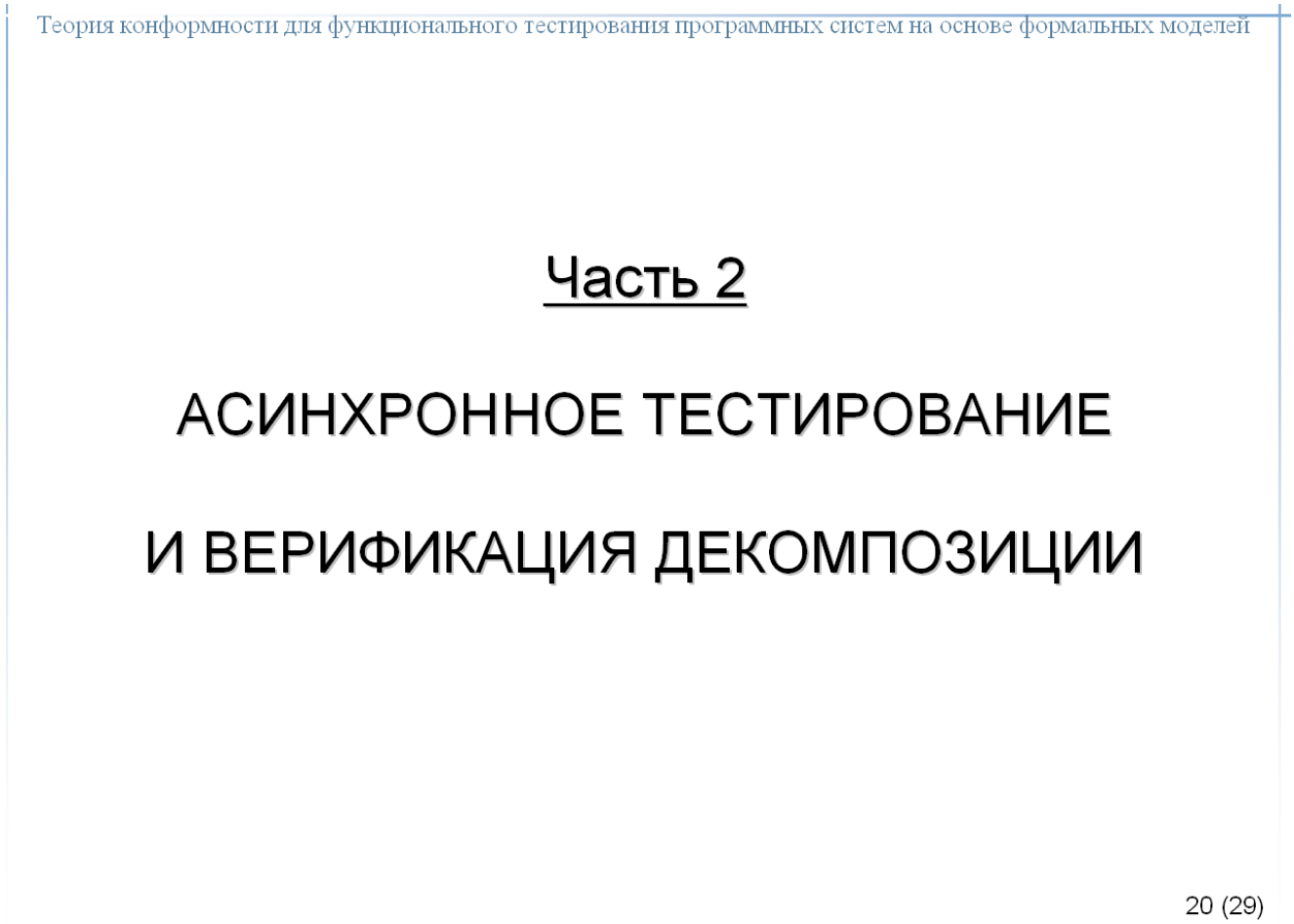


(теорема_16) Если набор трассовых тестов полон, то набор соответствующих LTS-тестов тоже полон.

Игорь Борисович Бурдонов, ИСП РАН

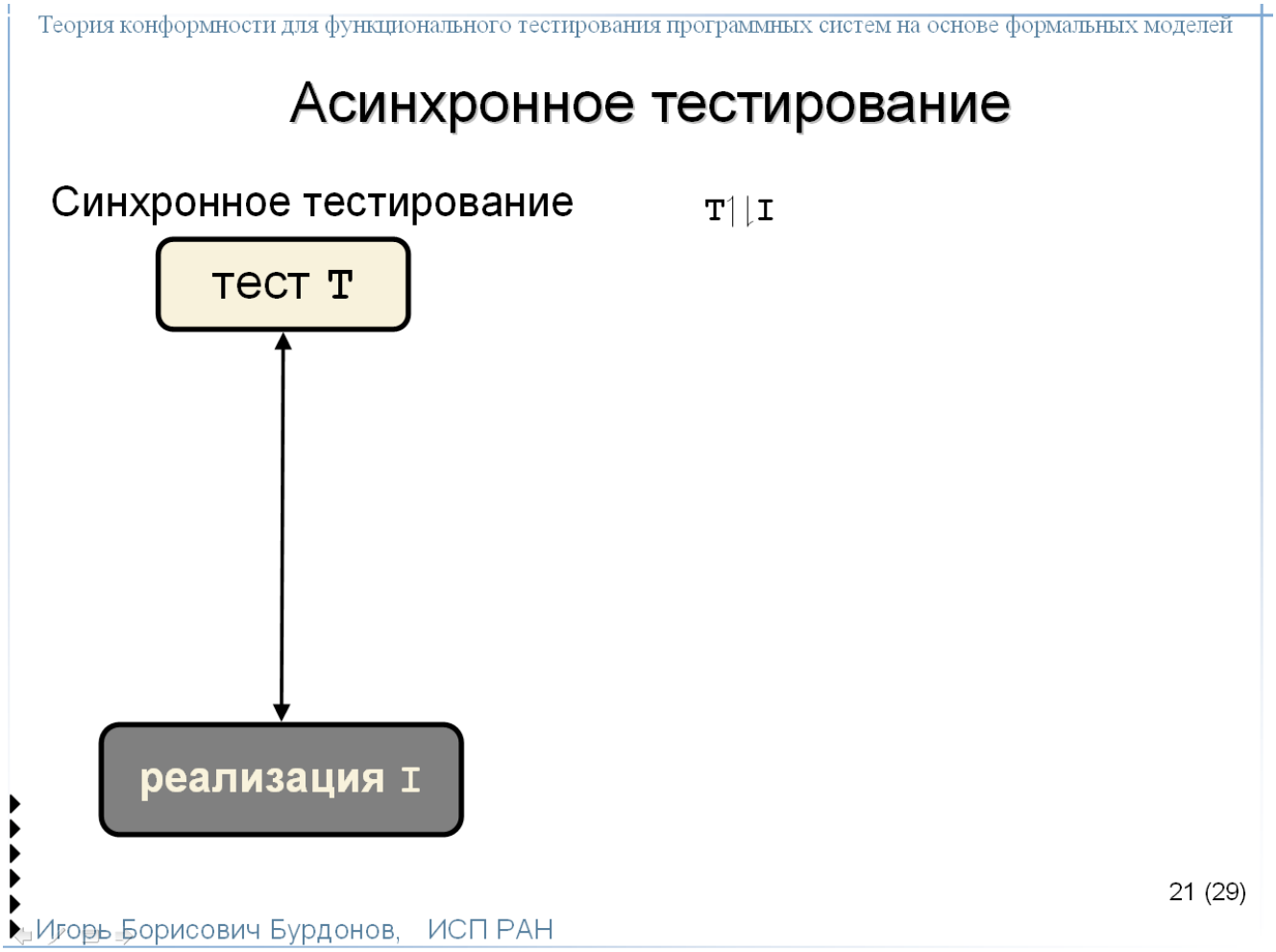
Действия заменяются на противоположные, а отказы – на символ θ.

20 слайд.



Теперь переходим к проблемам асинхронного тестирования и верификации декомпозиции.

21 слайд.



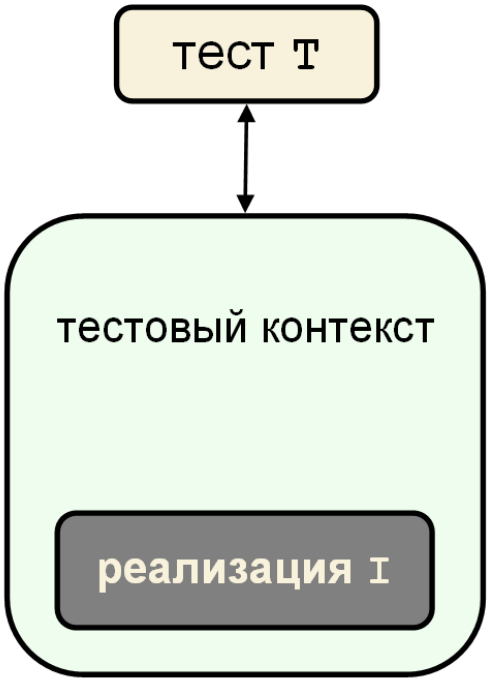
До сих пор рассматривалось синхронное тестирование, когда взаимодействие теста и реализации моделируется оператором параллельной композиции CCS.

Асинхронное тестирование

Асинхронное тестирование

$T \mid\mid I$

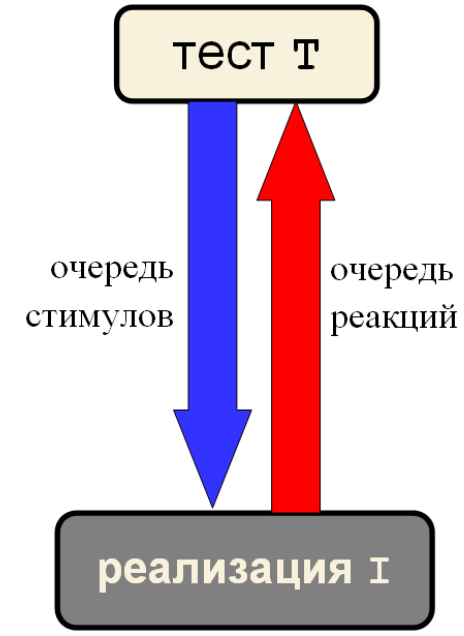
$T \mid\mid Context(I)$



В стандарте ISO асинхронное тестирование определяется как тестирование реализации, помещённой в тестовый контекст.

Асинхронное тестирование

Асинхронное тестирование



$T \lfloor\!\rfloor I$

$T \lfloor\!\rfloor Context(I)$

Сериализация: Асинхронной трассе $Context(I)$ соответствует множество синхронных трасс I. Если ни одной из них нет в спецификации S, то это ошибка.

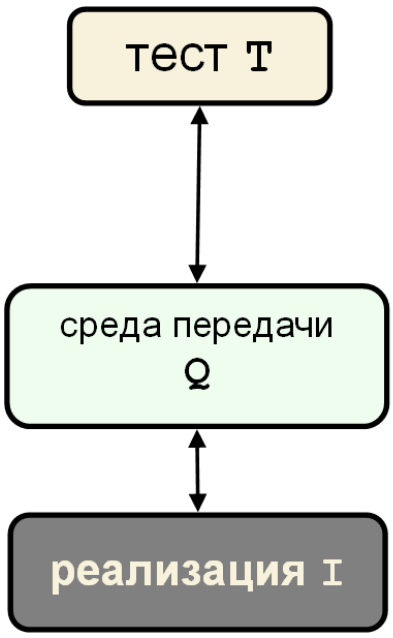
Безопасность: Блокировок стимулов нет, но дивергенция и разрушение остаются.

Игорь Борисович Бурдонов, ИСП РАН

Чаще всего такой контекст понимают как пару неограниченных очередей, буферизующих стимулы и реакции. Чтобы правильно оценивать результаты тестирования, мы должны понимать, что в реализации проходится не обязательно та трасса, которую мы наблюдаем, как это имело место при синхронном тестировании. Наблюдаемой трассе соответствует, вообще говоря, множество синхронных трасс, которые могла бы пройти реализация при таком наблюдении. Ошибка фиксируется, если ни одной из этих трасс нет в спецификации. Этот процесс называется сериализацией.

Асинхронное тестирование

Асинхронное тестирование



$T \| I$ $\quad Context(I) = I \| Q$

$T \| Context(I) = T \| (I \| Q)$

Сериализация: Асинхронной трассе $Context(I)$? соответствует множество синхронных трасс I. Если ни одной из них нет в спецификации S, то это ошибка.

? Безопасность: Блокировок стимулов нет, но дивергенция и разрушение остаются.

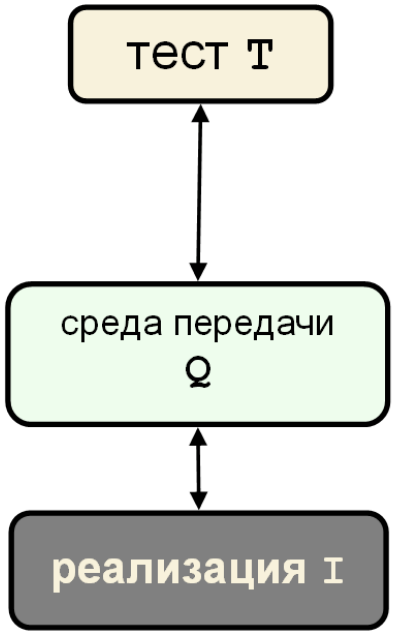
Игорь Борисович Бурдонов, ИСП РАН

В общем случае тестовый контекст можно понимать как среду передачи, которая моделируется подходящей LTS и компонуется с реализацией. Это может быть несколько входных и выходных очередей, очереди с приоритетами, стеки, порядок передачи может нарушаться, стимулы и реакции могут пропадать или генерироваться лишние и так далее.

Для каждой такой среды определение безопасных тестовых воздействий и сериализацию нужно делать по своим правилам. Составление правил такого тестирования – это интеллектуальный труд, далеко не всегда простой.

Асинхронное тестирование

Асинхронное тестирование



$T \mid\mid I \qquad Context(I) = I \mid\mid Q$

$T \mid\mid Context(I) = T \mid\mid (I \mid\mid Q)$

Решение для произвольной среды передачи:
алгоритм построения композиции $S \mid\mid Q$
и генерация тестов по композиции $S \mid\mid Q$.

Проблема несохранения конформности
(асинхронный тест ловит ложные ошибки)

$I \; saco \; S \not\Rightarrow I \mid\mid Q \; saco \; S \mid\mid Q$

Предлагаемое решение проблемы:
преобразование $\mathcal{T}$

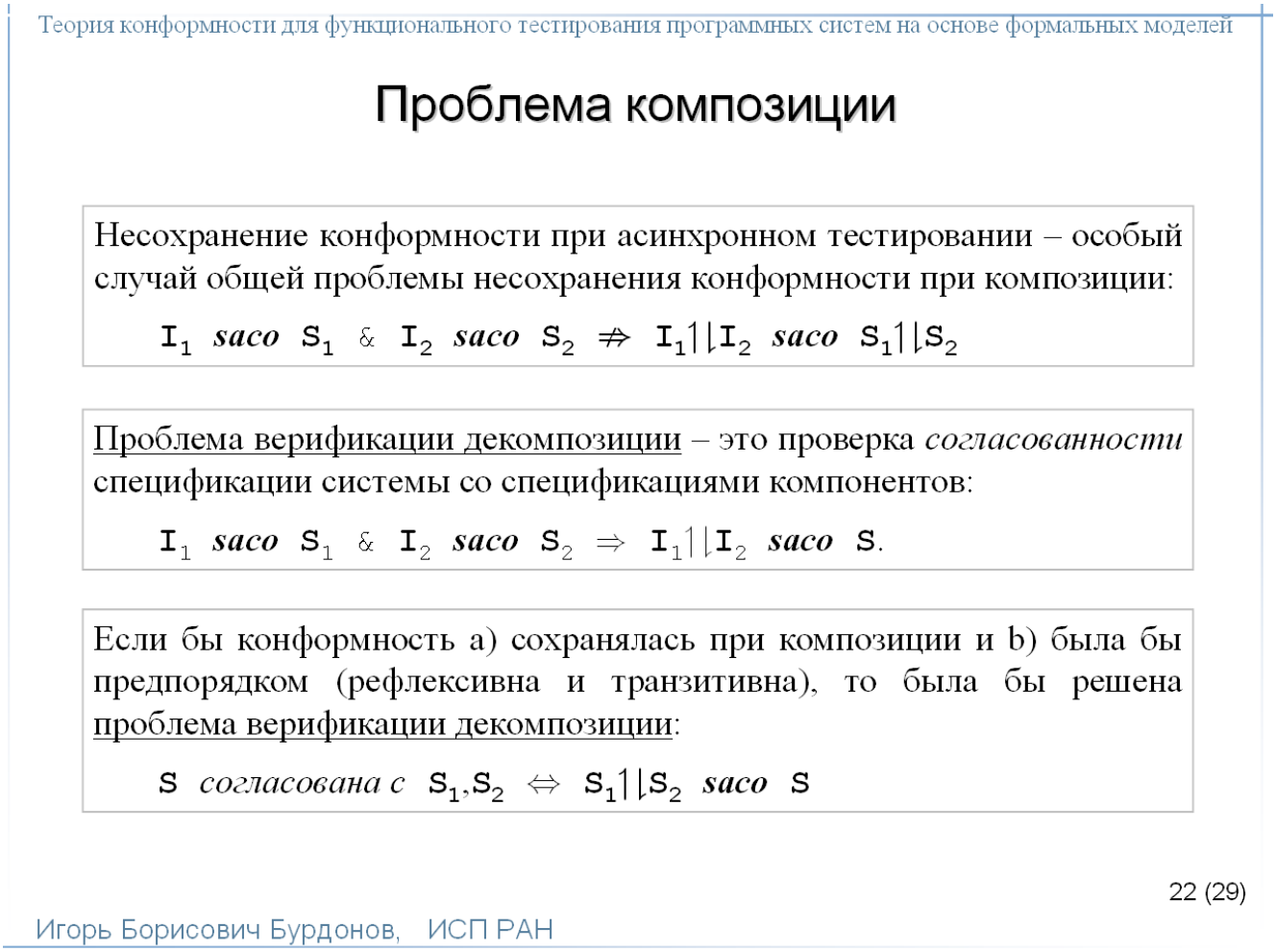
$I \; saco \; S \Rightarrow I \mid\mid Q \; saco \; \mathcal{T}(S) \mid\mid Q$

Решением является общий метод, когда тесты генерируются не по исходной спецификации с учётом этих правил, а по композиции спецификации со средой, и по ней же проверяются наблюдения.

Однако здесь возникает, так называемая, проблема несохранения конформности. Дело в том, что асинхронные тесты могут ловить ложные ошибки.

Для решения этой проблемы в работе предлагается специальное преобразование спецификации такое, чтобы для преобразованной спецификации конформность сохранялась при асинхронном тестировании. Разумеется, преобразование само должно сохранять класс конформных реализаций. Также желательно, чтобы оно не сужало класс безопасно-тестируемых реализаций.

22 слайд.



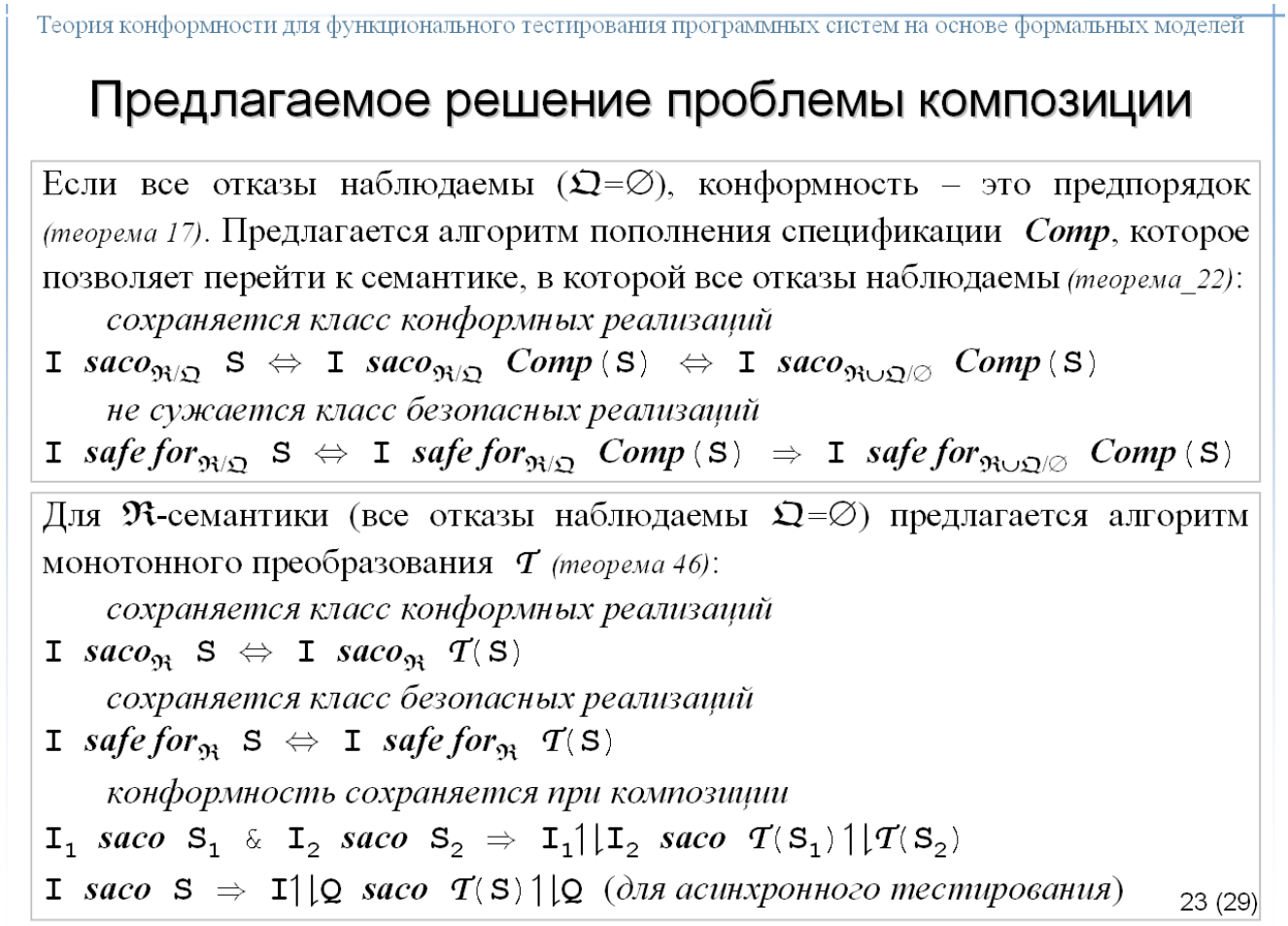
Несохранение конформности при асинхронном тестировании – это особый случай общей проблемы несохранения конформности при композиции. Суть этой проблемы в том, что если реализации компонентов составной системы конформны своим спецификациям, то отсюда вовсе не обязательно следует, что система, собранная из таких реализаций, будет конформна композиции спецификаций компонентов.

С этой проблемой тесно связана проблема верификации декомпозиции, то есть правильно ли спецификация составной системы разложена на спецификации её компонентов. Спецификации системы и компонентов согласованы, если система, собранная из любых конформных реализаций компонентов, оказывается конформна спецификации системы.

Если отношение конформности a) сохраняется при композиции и b) является предпорядком (рефлексивно и транзитивно), то

проблема верификации декомпозиции сводится к проверке того, что композиция спецификаций компонентов конформна спецификации системы.

23 слайд.

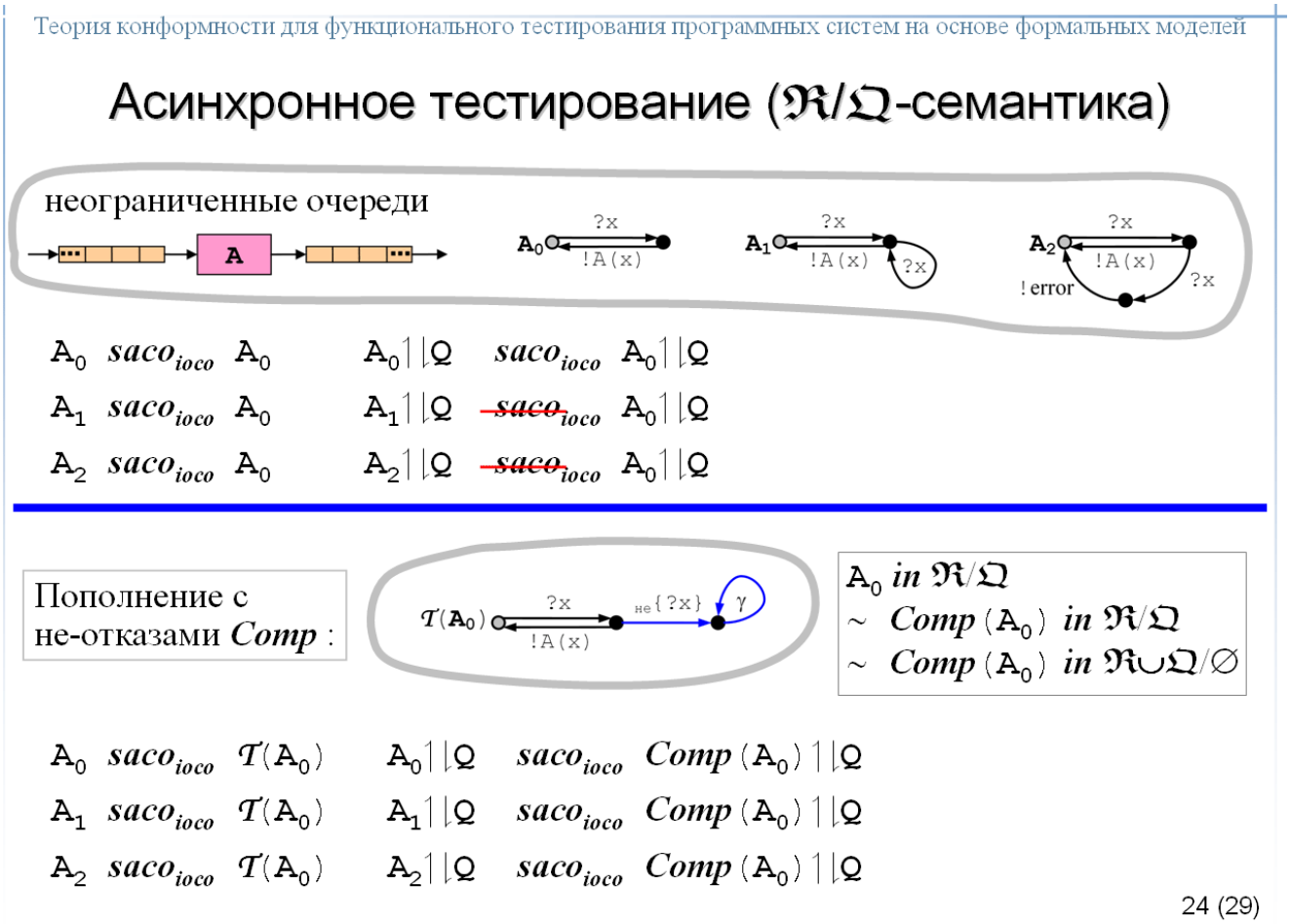


Поэтому предлагаемое в работе решение проблемы композиции сводится к двум преобразованиям спецификации: пополнению и монотонному преобразованию.

В диссертации доказано, что, если в семантике все отказы наблюдаемы, то конформность является предпорядком. Пополнение – это переход к такой семантике. Сохраняется класс конформных реализаций как в исходной, так и в новой семантиках, а класс безопасных реализаций не сужается.

Монотонное преобразование применяется уже в рамках семантики без ненаблюдаемых отказов. Сохраняется как класс конформных, так и класс безопасных реализаций. Для монотонно преобразованных спецификаций конформность сохраняется при композиции, а также при асинхронном тестировании.

24 слайд.



24 (29)

Теперь я хочу продемонстрировать как эти проблемы, так и предлагаемые решения, на нескольких примерах.

Первый пример – это система A, которую мы рассматривали для ***ioco***-семантики. Мы видели, что все три изображённые на слайде реализации безопасно-конформны.

Что произойдёт при асинхронном тестировании, когда среда передачи – это неограниченные входная и выходная очереди?

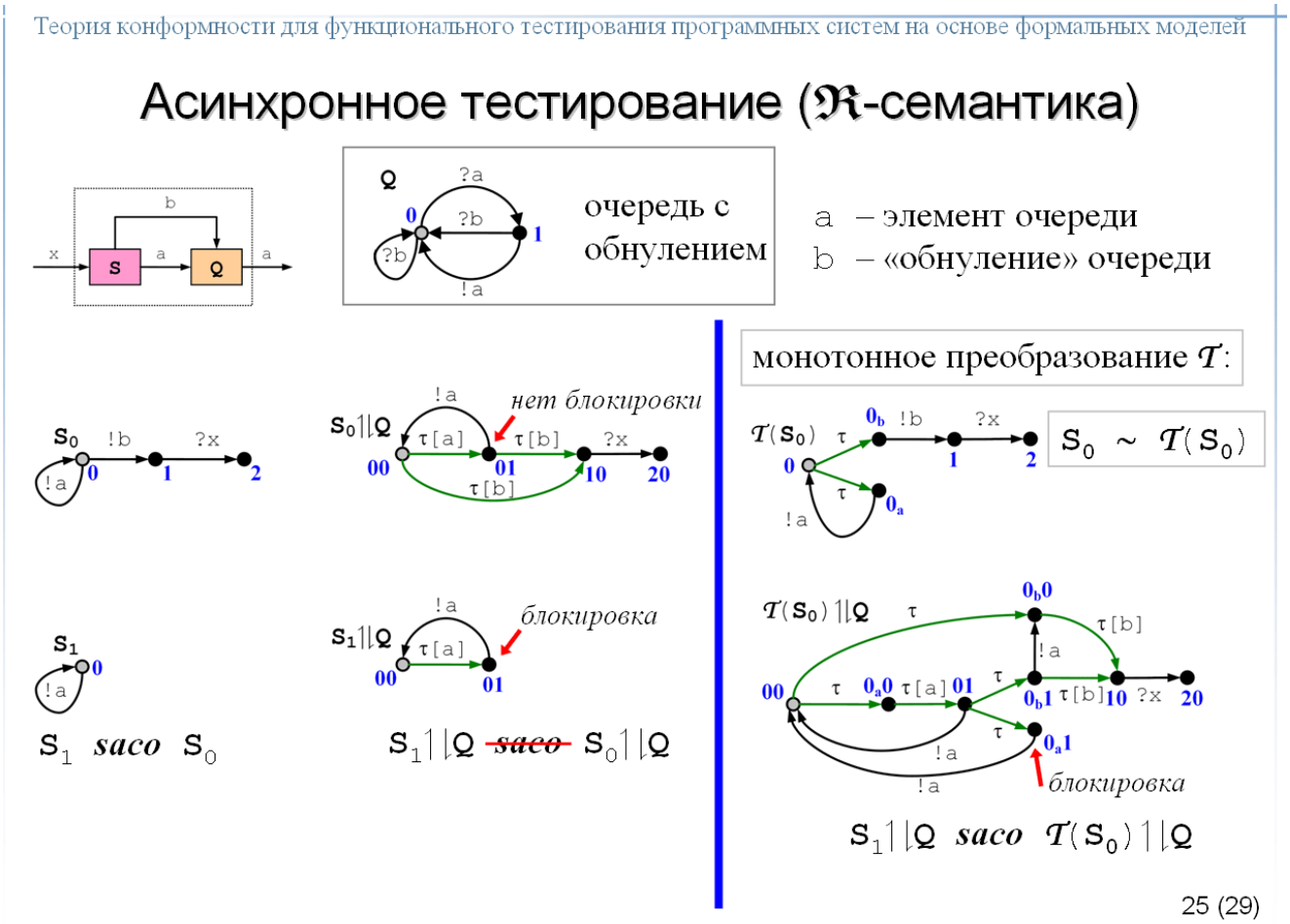
Теперь нам ничто не мешает посылать из теста несколько стимулов подряд: входная очередь все их примет и буферизует, а реализация будет потом выбирать эти стимулы из очереди. Если послать два стимула x_1 и x_2, а потом ждать реакций, то, согласно спецификации, должно быть две реакции $A(x_1)$ и $A(x_2)$. В то же время для реализаций A_1 и A_2 возможен и

другой вариант: только одна реакция: правильная $A(x_1)$ или даже ошибочная ***error***.

Корень проблемы в том, что спецификация не всюду определена по стимулам, а блокировки стимулов не наблюдаемы. Решение заключается в том, чтобы избавиться в спецификации от ненаблюдаемых отказов. Введём фиктивное действие «не-блокировка x» и будем считать, что, посылая стимул x, мы разрешаем реализации выполнять также и это действие. В спецификации там, где стимул x не принимается, добавим переход по «не-блокировке x» и далее разрушение. Тогда при безопасном асинхронном тестировании мы просто не имеем права давать второй стимул до получения реакции, так как это может привести к разрушению.

В диссертации дано определение общего преобразования пополнения и доказана его применимость к любым LTS-спецификациям и $\mathfrak{R}/\mathfrak{Q}$-семантикам.

25 слайд.



Пополнение спецификации решает первую задачу – избавляет нас от ненаблюдаемых отказов. Но проблема несохранения конформности остаётся и для 𝔑-семантик, в которых все отказы наблюдаемы.

На слайде изображён пример асинхронного тестирования системы S с помощью среды передачи Q. Среда – это одна ограниченная выходная очередь с дополнительной командой «обнуления». Здесь очередь имеет длину 1.

Спецификация S_0 рассматривается в семантике с наблюдаемыми блокировками. Она описывает следующие требования к системе: сначала стимул блокируется, но можно выдавать цепочку реакций a, потом можно обнулить очередь, принять стимул x и закончить в терминальном состоянии.

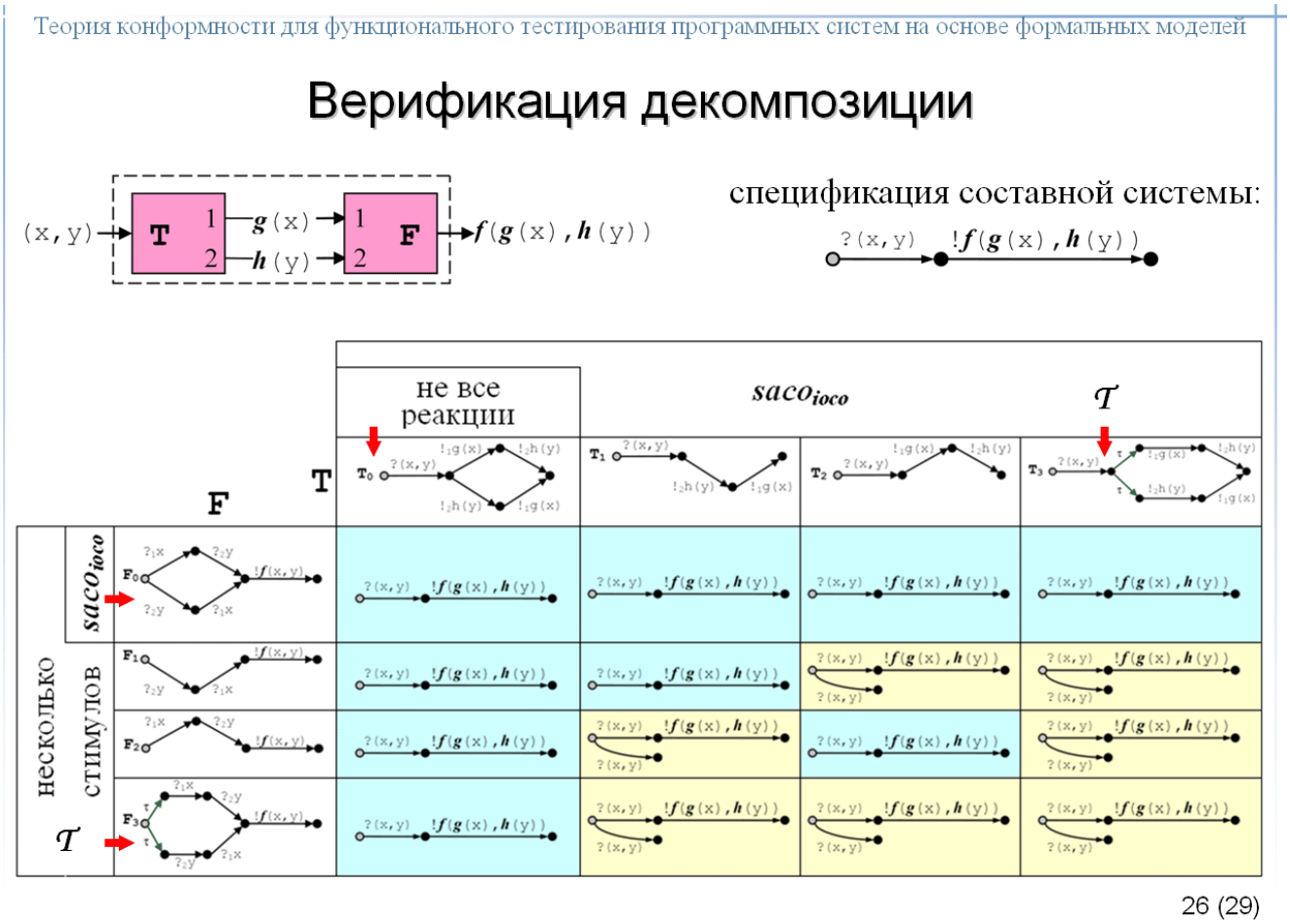
Реализация S_1 конформна: просто она не обнуляет очередь и, соответственно, не принимает стимул.

Когда с очередью Q компонуется спецификация, первый посылаемый стимул не блокируется. Однако при композиции реализации S_1 такая блокировка появляется, что при асинхронном тестировании будет квалифицировано как ошибка. Это показано красными стрелками.

Монотонное преобразование, которое здесь требуется, должно «расщепить» начальное состояние 0, добавив состояния $0a$ и $0b$. В одном состоянии выдаётся только реакция a, а в другом – только реакция b. За счёт этого в композиции спецификации с очередью блокировка первого стимула становится возможной, и композиция реализации S_1 с очередью теперь конформна.

В диссертации дано определение общего монотонного преобразования и доказана его применимость к любым LTS-спецификациям и $\mathfrak{R}$-семантикам.

26 слайд.



Это пример верификации декомпозиции. Здесь система составлена из двух компонентов: T и F, которые мы рассматривали как примеры для приёма части реакций и посылки нескольких стимулов. Спецификация составной системы очень проста: нужно принять пару аргументов (x,y), и выдать значение ***f***(***g***(x),***h***(y)). Интуитивно именно это и должны делать в совокупности реализации компонентов: компонент T со спецификаций T_0 вычисляет функции ***g*** и ***h***, а компонент F со спецификаций F_0 вычисляет функцию ***f***.

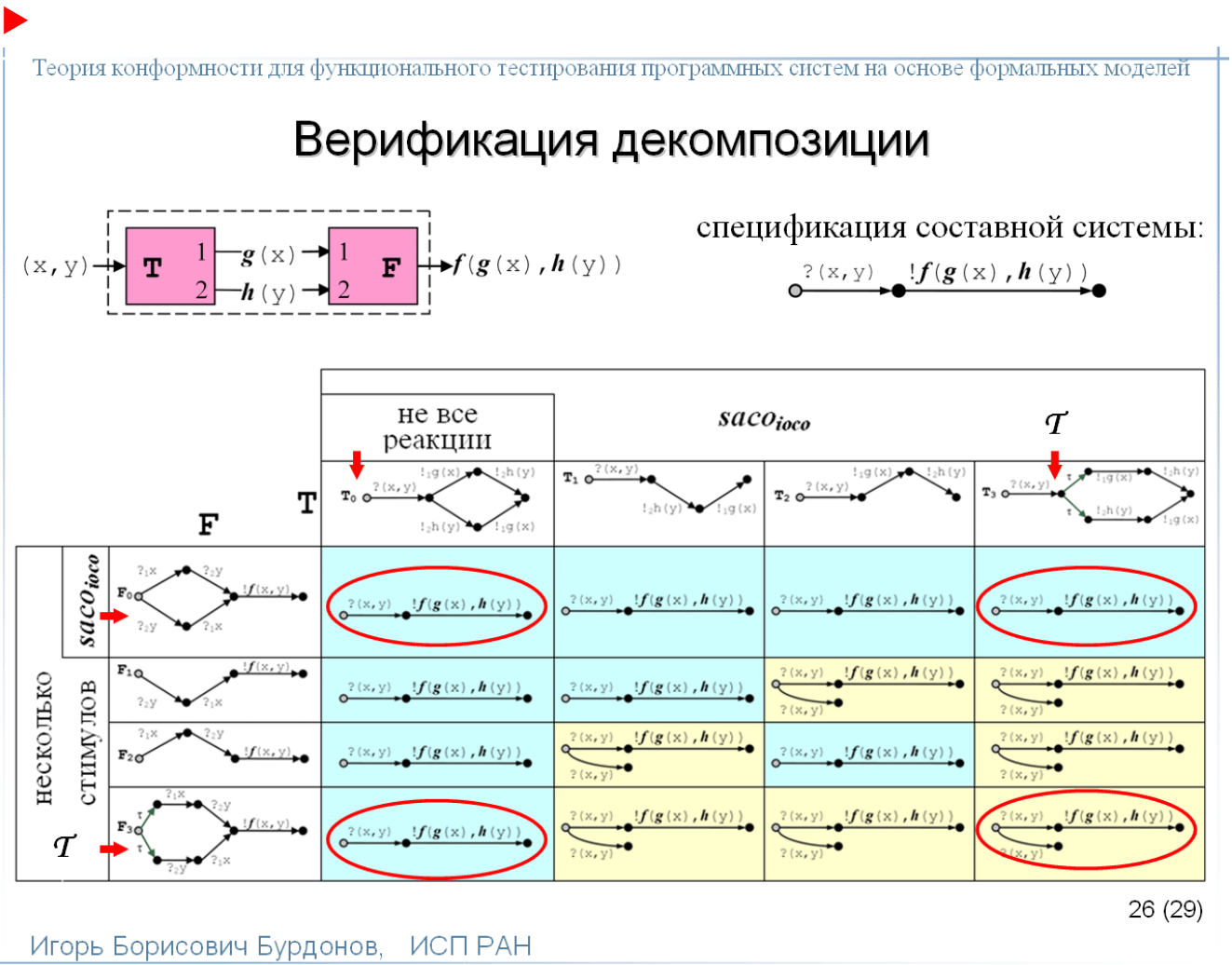
В таблице показаны спецификации компонентов и ещё по три реализации. Каждый компонент рассматривается в двух семантиках: в одной семантике конформна только сама спецификация, в другой – все реализации. Голубые клетки соответствуют случаям, когда композиция реализаций

компонентов конформна спецификации системы, а жёлтые клетки – когда это не так: в композиции реализаций появляется состояние, в котором реакция не выдаётся.

Мы видим, что не всякое сочетание семантик компонентов T и F гарантирует согласованность спецификаций компонентов со спецификацией системы. Из четырёх вариантов, одно ошибочное, а три правильных. Раньше мы рассматривали только одно правильное сочетание.

Как же верифицировать декомпозицию?

На слайде для каждого компонента красными стрелками показаны монотонно преобразованные спецификации в зависимости от семантики, в которой рассматривалась исходная спецификация.

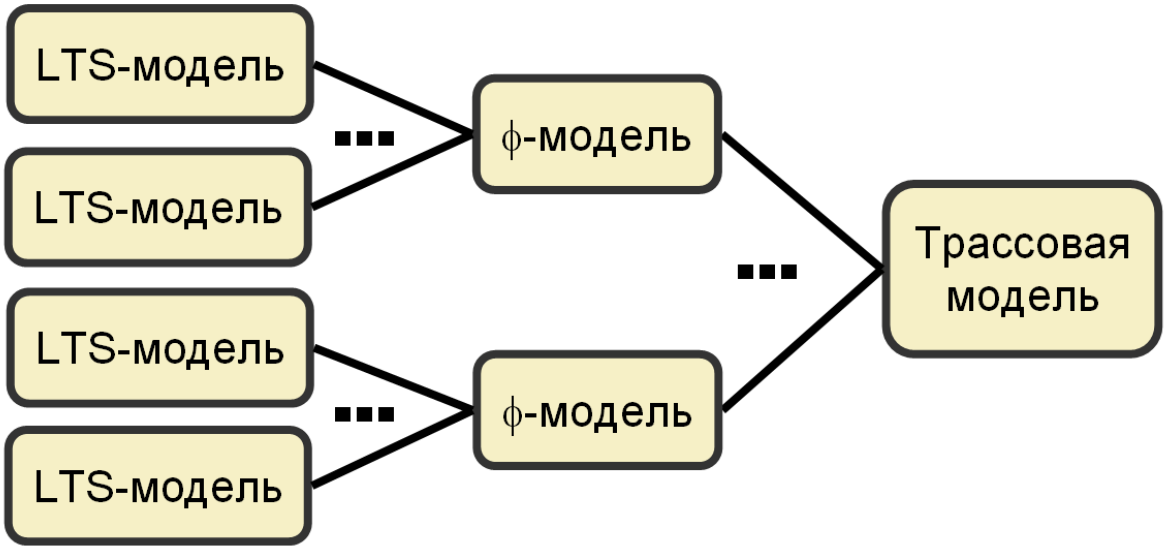


Вот теперь я отмечаю красными овалами те клетки таблицы, в

которых нарисована как раз композиция монотонно преобразованных спецификаций. В голубых клетках получается композиция, которая конформна заданной спецификации, а в одной жёлтой клетке – неконформная композиция. В этом последнем случае спецификации компонентов и системы не согласованы.

27 слайд.

ϕ-модель как средний уровень абстракции



1. Трассовая модель однозначно определяется ϕ-моделью.
2. Можно определить композицию ϕ-моделей:
   ϕ-модель композиции LTS = композиции ϕ-моделей LTS.

Игорь Борисович Бурдонов, ИСП РАН

Для построения теории монотонных преобразований в диссертации введена новая модель, названная ϕ-моделью, которая занимает промежуточный уровень абстракции между моделью наблюдаемых трасс и LTS-моделью.

Во-первых, ϕ-модель однозначно определяет трассы наблюдений. Тем самым, у нас есть всё для определения гипотезы о безопасности, безопасной конформности и генерации тестов.

Во-вторых, для ϕ-моделей определяется композиция таким образом, что ϕ-модель композиции LTS совпадает с композицией ϕ-моделей этих LTS.

Совокупность этих двух свойств как раз и обеспечивает успех в решении проблемы композиции.

В диссертации ϕ-модель определяется как множество трасс, обладающее определённым набором свойств. Но только это не трассы наблюдения в $\mathfrak{R}/\mathfrak{Q}$-семантике: вместо наблюдаемых отказов там используется более полная информация о состоянии в терминах действий, по которым есть переходы из этого состояния.

Конечно, ϕ-теория – это «леса», которые нужны для построения монотонного преобразования. Но, мне кажется, эти леса могут быть полезны и сами по себе.

28 слайд.

Тестирование в ИСП РАН

(контрактные спецификации, безопасное тестирование с учётом разрушения)

Модель	Тестирование	Контекст	$\mathfrak{R}$	Ω	Метод	Ограничения	Инструмент	Проекты
Автомат	Синхронное	–	$\varnothing$	Блокировки стимулов δ	Обход графа	Детерминированный граф сильно-связный	KVEST UniTESK	1
					Обход графа по стимулам	Недетерминиров. граф с сильно-связным полным остовным детерминиров. подграфом	UniTESK	2
						Недетерминиров. граф сильно-Δ-связный	[UniTESK]	–
Автомат /LTS	Асинхронное	Неограниченные входные очереди и выходные очереди 1,2,… + Непосредственные реакции 0			Вызов из параллельных процессов	Дополнение к синхронному тестированию	KVEST	3
LTS			δ_1 δ_2 …	Блокировки стимулов δ_0	Стационарное тестирование (обход стационарн. графа)	В стационарном состоянии все стимулы принимаются (возможно, с разрушением)	KVEST	4
							UniTESK	5
	Синхронное/ Асинхронное				Нестационарное тестирование	Дополнение к стационарн. тестированию	[UniTESK]	6

№	Название проекта	Годы	Заказчик
1	Kernel Verification	1994-8	Nortel
	FPE	1998-9	Nortel
	XA-Core	2000	Nortel
2	OLVER	2005-6	Федеральное Агентство по науке и инновациями (Роснаука)
	OC 2000	2005-8	НИИСИ
	Java Infrastructure	2004	Intel
	Hardware Design Testing	2006-8	НИИСИ
	Germany Banking Software	2004-5	Luxoft
3	Kernel Verification	2000	Nortel
	XA-Core	2000	Nortel
	ORB	2000	Nortel
4	GWC	2000	Nortel
5	MSR IPv6	2000-1	Microsoft Research
	Mobile IPv6	2001-2	Microsoft Research, РФФИ
	OLVER	2005-6	Федеральное Агентство по науке и инновациями (Роснаука)
	OC 2000	2005-8	НИИСИ
	Hardware Design Testing	2006-8	НИИСИ
6	Верификация распределенных систем	2003-5	Программа Президиума РАН «Разработка фундаментальных основ создания научной распределённой информационно-вычислительной среды на основе технологий GRID»
	Проблемно-ориентированные методы автоматизированной верификации распределённых систем	2006-8	

Здесь показаны те методы, которые мы применяли в системах тестирования KVEST и UniTESK для указанных реальных проектов. Мы писали спецификации в пред- и постусловиях. Из них извлекались вручную или автоматическим способом модели в виде автоматов или LTS. Под разрушением понималось поведение программы после обращения к ней с нарушением её предусловия. Это учитывалось при тестировании, то есть оно было безопасным.

Таблица показывает три вещи.

Во-первых, теория строилась как осмысление и обобщение того практического тестирования на основе формальных моделей, которое мы проводили.

Во-вторых, построенная теория покрывает все эти частные случаи. Конечно, общий метод генерации тестов, описанный в

работе, в каждом из этих практических случаев оптимизирован с учётом указанных ограничений на реализации и спецификации.

В-третьих, то, ради чего теория создавалась. Она показывает пути решения проблем тестирования для других семантик взаимодействия и других контекстов при асинхронном тестировании. Некоторые из них я продемонстрировал раньше на простейших примерах.

29 слайд.

Основные результаты работы

1. **Метод формализации тестового эксперимента** с помощью параметризуемой машины тестирования для определения класса семантик взаимодействия. Введение понятия *разрушения*.
2. **Введение концепции безопасного тестирования, гипотезы о безопасности и безопасной конформности**, параметризуемой семантикой взаимодействия.
3. **Метод генерации полного набора тестов для безопасной конформности** по трассовой или LTS-спецификации в заданной семантике взаимодействия.
4. **Метод пополнения спецификаций**, сохраняющего класс конформных и не сужающий класс безопасных реализаций, для перехода к семантике, в которой все отказы наблюдаемы. Решение проблемы рефлексивности и транзитивности конформности.
5. **Разработка концепции ϕ-трасс и модели ϕ-трасс**, объединяющей конформность и композицию моделей.
6. **Метод монотонного преобразования спецификаций**, с помощью которого решаются проблемы сохранения конформности при асинхронном тестировании и автоматической верификации декомпозиции системных требований.

Игорь Борисович Бурдонов, ИСП РАН

Здесь перечисляются основные результаты работы.

Итак, удалось построить теорию конформности для $\mathfrak{R}/\mathfrak{Q}$-семантик, для которой все практически применяемые семантики взаимодействия рассматриваемого класса являются частными случаями.

Введено понятие разрушения, моделирующего поведение, которое по тем или иным причинам не должно появляться при тестировании.

Предложена концепция безопасного тестирования, учитывающая разрушение и расширяющая классы тестируемых и конформных реализаций.

Заложены основы алгоритмической генерации тестов, позволяющие для любой заданной $\mathfrak{R}/\mathfrak{Q}$-семантики строить

инструменты генерации тестов, которые уже могут учитывать те или иные ограничения на классы спецификаций и реализаций.

Определены пополнение и монотонное преобразование, при помощи которых впервые в полном объёме решаются задачи асинхронного тестирования и верификации декомпозиции спецификаций.