

Верификация параллельных программных и аппаратных систем



Курс лекций

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского политехнического университета
karpov@dcn.infos.ru



План курса

1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Исчисление взаимодействующих систем (CCS) Р.Милнера
4. Темпоральные логики
5. Алгоритм model checking для проверки формул CTL
6. Автоматный подход к проверке выполнения формул LTL
7. Структура Крипке как модель реагирующих систем
8. Темпоральные свойства систем
9. Система верификации Spin и язык Promela. Примеры верификации
10. Применения метода верификации model checking
11. BDD и их применение
12. Символьная проверка моделей
13. Количественный анализ дискретных систем при их верификации
14. Верификация систем реального времени (I)
15. Верификация систем реального времени (II)
16. Консультации по курсовой работе



Лекция 6

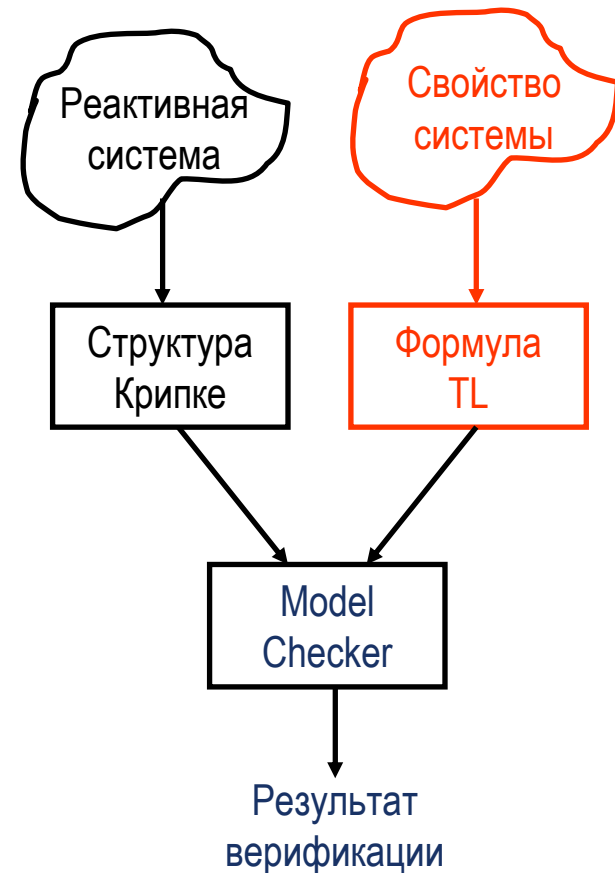
Темпоральные свойства систем

Примеры проверяемых свойств

В основном, для каждой системы нужно придумывать свои требования

Если это свойство LTL, то требуется его выполнение на каждом вычислении

- $EF (\text{Start} \wedge \neg \text{Ready})$
можно попасть в состояние, в котором система стартовала, но не готова
- $GF \text{ DeviceEnabled}$
условие *DeviceEnabled* выполняется неопределенно часто
- $G[q \Rightarrow X [(\neg p \vee G \neg r)]]$
Между q и r свойство p никогда не выполнится





Спецификация причинной зависимости

Попытка выразить свойство P:

“запрос req в конце концов приведет к получению ack”

- $req \Rightarrow ack$

неправильно: *“в начальном состоянии если req истинно, то истинно и ack”*

- $Greq \Rightarrow ack.$

Неправильно: понимается как $(Greq) \Rightarrow ack$, *“если во всех состояниях выполняется req, то в начальном выполняется ack”*

- $G(req \Rightarrow ack)$

Неправильно: *“в любом состоянии если выполняется req, то в этом же состоянии должно выполняться и ack”*

Спецификация причинной зависимости (2)

Попытка выразить свойство P:

"запрос req в конце концов приведет к получению ack"

- **G(req \Rightarrow Fack)**

Не совсем правильно: "в любом состоянии если выполняется req, то когда-нибудь в будущем (включая настоящее) должно выполняться и ack". Эта формула получила свой значок " $\sim>$ ": $\phi \sim> \psi \equiv \mathbf{G}(\phi \Rightarrow \mathbf{F} \psi)$. Но она будет истинна и в случаях, когда в том же состоянии, в котором истинно req, будет истинно и ack

- **G(req \Rightarrow XFack)**

более точно отражает причинно-следственную связь между req и ack, но выполняется и в случае, если req не наступает!

- **G(Freq & (req \Rightarrow XFack))**

наиболее точно отражает идею причинно-следственной связи событий req и ack – "событие req происходит бесконечно часто, и, если это событие произошло, то потом произойдет и событие ack"



Примеры проверяемых свойств (LTL)

- После того, как сигнал p стал активным, сигнал q не будет активным до тех пор, пока не станет активным r

$$G[(p \Rightarrow X(\neg q \cup r))]$$

- После того, как сигнал p стал активным, сигнал q будет активным по крайней мере три шага

$$G[p \Rightarrow Xq \ \& \ XXq \ \& \ XXXq]$$

- Между q и r свойство p никогда не выполнится

$$G[q \Rightarrow X[(\neg p \cup r) \vee G \neg r]]$$



Примеры проверяемых свойств (LTL)

Частичная корректность

$$(\text{at_Start} \wedge \varphi) \Rightarrow G (\text{at_Finish} \Rightarrow \psi)$$

Локальный инвариант (в состоянии s выполняется свойство)

$$G (\text{at_s} \Rightarrow \varphi)$$

Взаимное исключение:

$$G (\neg(\text{at_crint1} \wedge \text{at_crint2}))$$

Отсутствие взаимной блокировки:

$$G (\text{enabled}_1 \vee \dots \vee \text{enabled}_n)$$

Запрос не будет снят, пока на него не придет подтверждение:

$$G (\text{req} \Rightarrow (\text{req} \text{ U } \text{ack}))$$

Если система пожаротушения включена, то на это предварительно была получена санкция капитана:

$$G (\text{firefighting} \Rightarrow P \text{ CaptainAckFF})$$



Требования к системе выделения ресурсов (LTL)

- Два процесса запрашивают неразделяемый ресурс. Требования к системе выделения ресурсов:
 - никогда оба процесса не будут владеть ресурсом одновременно
 $G \neg(\text{owns1} \wedge \text{owns2})$
 - если первый процесс запросит ресурс, то он его получит ($\text{req1} \leadsto \text{owns1}$)
 $G (\text{req1} \Rightarrow F \text{owns1})$
 - если процесс 1 запрашивает ресурс неопределенно часто, он его получит
 $GF (\text{req1} \wedge \neg (\text{owns1} \vee \text{owns2})) \Rightarrow GF \text{owns1}$



Примеры проверяемых свойств (CTL)

- $EF(\text{Start} \ \& \ \neg \text{Ready})$

- достижимо состояние, в котором свойство Start выполняется, а Ready – нет

- AGAF Restart

- при любом функционировании системы (на любом пути) из любого состояния системы всегда обязательно вернемся в состояние рестарта

- AG EF Restart

- при любом функционировании системы (на любом пути) из любого состояния системы существует путь, по которому можно перейти в состояние рестарта

- $E[p \ U \ A [q \ U \ r]]$

- существует путь, на котором p выполняется до тех пор, пока в будущем q будет всегда выполняться до выполнения r



Классификация свойств

Существуют и общие свойства, типичные для многих реактивных систем

Множество типичных свойств реактивных систем разделено на (пересекающиеся) классы

Важность выделения классов свойств в том, что при проектировании системы следует проверять кроме специфичных, еще и общие, свойства, позволяющие проверить отсутствие типичных некорректностей параллельных систем



Классы свойств реактивных систем

- Достижимость (*reachability*)
- Безопасность (*safety*) - *нечто плохое никогда не произойдет*
- Свобода от дедлоков (*deadlock freeness*) – проверка блокировок
- Живость (*liveness*) - *нечто хорошее обязательно произойдет*
- Справедливость (*fairness*) – дополнительные ограничения

Достижимость (*reachability*)

Некоторая ситуация может быть достигнута при функционировании

- Простое требование достижимости
 - R1: параметр N станет отрицательным
 - R2: система войдет в критическую секцию
- Вариант – отрицание некоторого свойства
 - R3: параметр N никогда не станет отрицательным
 - R4: процесс никогда не перейдет в состояние crash

Естественным образом свойство достижимости выражает формула $EF\varphi$, где φ не содержит темпоральных операторов:

- R1: $EF\ N < 0$
- R2: $EF\ crit_section$
- R3: $\neg EF\ N < 0$
или $AG\ \neg N < 0$
- R4: $\neg EF\ crash$
или $AG\ \neg crash$

Проверка простого свойства достижимости – построением пространства достижимых состояний. Это – самая простая задача верификации || систем



Условная достижимость

Достижимость
Безопасность
Дедлоки
Живость
Справедливость

- Требование достижимости может сопровождаться условиями
 - R5: система войдет в критическую секцию без прохождения $N=0$
- Например, относительно некоторого состояния системы
 - R6: система *всегда* может вернуться в исходное состояние (*из любого достижимого*)
 - R7: система может вернуться в исходное состояние (*из текущего состояния системы*)

Условная достижимость требует Until:

- R5: $E[(N < 0) \text{ U } \text{crit}]$

Достижимость из любого состояния требует вложения EF в AG:

- R6: $AG \text{ EF init}$

Достижимость из текущего состояния (не означает достижимость из следующего):

- R7: $EF \text{ init}$



Безопасность (*safety*)

Nothing bad happens

Достижимость
Безопасность
Дедлоки
Живость
Справедливость

При некоторых условиях некоторая ситуация никогда не может быть достигнута (гарантия того, что *нечто плохое никогда не произойдет*)

Свойство нарушается iff оно нарушается некоторым конечным префиксом

Свойство безопасности выражается формулой:

$AG \neg \phi$

Безопасность с дополнительным условием:

$AG \neg \phi W \phi$ (weak Until)

Пример. “Два процесса никогда не войдут в свои критические секции одновременно”:

$AG \neg (crint1 \wedge crint2)$

(если требование безопасности – без условия, то оно является и свойством достижимости: $AG \neg \phi$)

$AG (req \Rightarrow A(req \text{ U } ack))$

“req must hold until ack”

Пример. “Два процесса никогда не войдут в свои критические секции одновременно”:

$AG \neg (crint1 \wedge crint2)$

(если требование безопасности – без условия, то оно является и свойством достижимости: $AG \neg \phi$)

Пример. “Пока ключ зажигания не вставлен, машина не двинется”

$AG(\neg \text{start W key})$

W: weak Until: $pWq = pUq \vee Gp$

Если формула $AG(\neg \text{start U key})$, то ключ обязательно должен быть вставлен (этого требует формула при любом развитии событий)

Исторические переменные

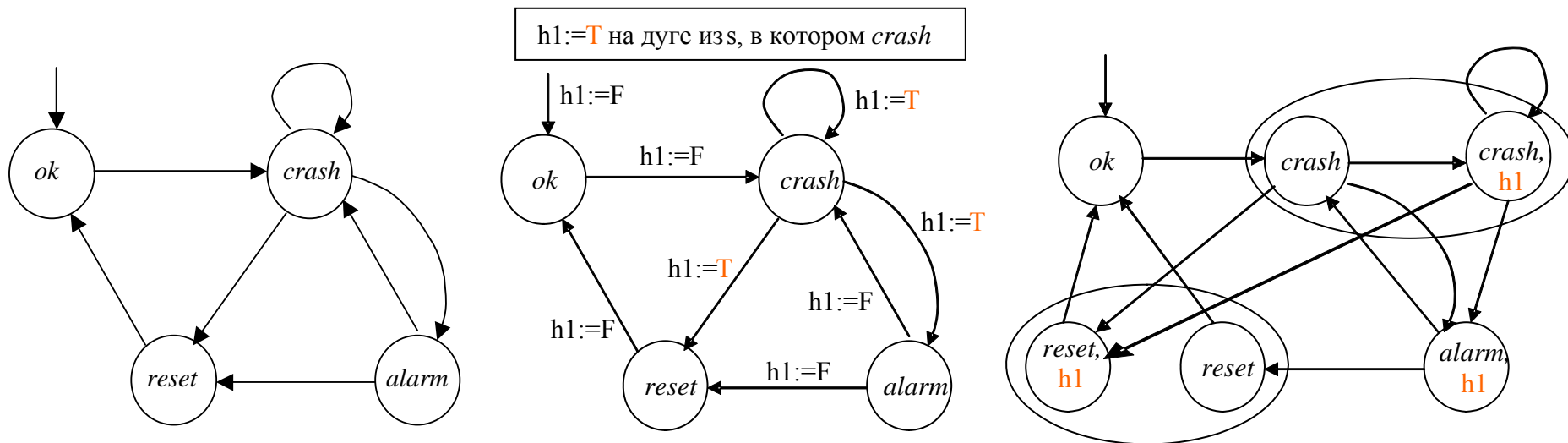
Достижимость
Безопасность
Дедлоки
Живость
Справедливость

Часто свойство безопасности выражается через операторы прошлого:
 Pq , $X^{-1}q$, pSq . P – обратное F (F^{-1}), X^{-1} обратное X , S – обратное U

Оператор $AG \phi^{-1}$ – типичная формализация безопасности с условием.

Пример: “сигнал ‘alarm’ установится только после того, как сигнал ‘crash’ появится в предыдущий момент времени”: $AG(alarm \Rightarrow X^{-1} crash)$

Как проверить K , не расширяя логику? Ведением исторических переменных



Вводим $h1 \equiv X^{-1}crash$, т.е. так, чтобы на развертке K $h1 \equiv T$ в s , если оператор прошлого в s выполняется. Тогда $AG(alarm \Rightarrow h1)$ – просто достижимость!

Исторические переменные (2)

Достижимость
Безопасность
Дедлоки
Живость
Справедливость

Пример: Каждый раз, когда появляется сигнал 'alarm', ему после последнего 'crash' не предшествует 'reset': $AG(alarm \Rightarrow (\neg reset) S crash)$

pSq : q *выполнилась в прошлом, и с того момента p все время истинно*

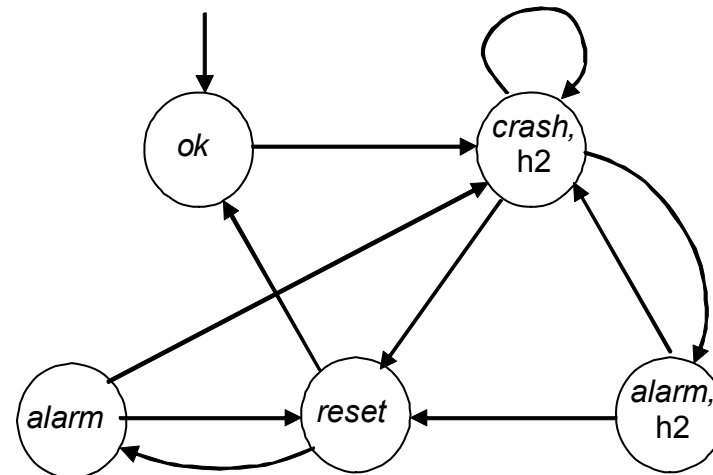
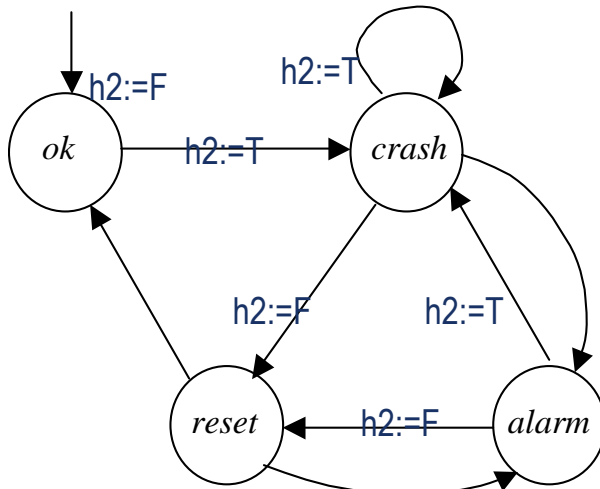
Введем историческую переменную $h2 \equiv (\neg reset) S crash$.

На начальной стрелке $h2 := F$

Далее, $h2 := T$ на каждом переходе, ведущем в состояние $crash$;

$h2 := F$ на каждом переходе, ведущем в состояние $reset$;

$h2$ не изменяется на каждом переходе, ведущем в состояние $\neg reset$



Это всегда можно сделать, поскольку эти истории конечны





Свобода от блокировок (дедлоков)

Достижимость
Безопасность
Дедлоки
Живость
Справедливость

В CTL обычно **AG EX true**: *Какое бы состояние ни было достигнуто (AG) существует непосредственный преемник этого состояния (EXtrue)*

Это не частный случай safety – безопасности, но для конкретной системы мы можем свести это к safety, если состояния дедлока опишем явно



Нечто при некоторых обстоятельствах когда-нибудь случится
(*нечто хорошее обязательно произойдет в будущем*)

Обычно живость характеризует прогресс в нужном направлении

L1: Любой запрос будет в конце концов обслужен

$AG(req \Rightarrow AFsat)$ в CTL, $G(req \Rightarrow Fsat)$ в LTL

L2: Если долго стараться, то в конце концов получится

L3: Если лифт вызван, он обязательно придет рано или поздно

L4: Светофор в конце концов переключится на зеленый

L5: После дождя выглянет, наконец, солнце

$G(\text{дождь} \Rightarrow A(\text{дождь} \cup \text{солнце}))$

L5' Солнце будет неопределенно часто

L6: Программа завершится (т.е. liveness не означает, что система бесконечно функционирует, отличается от живости Сети Петри)

L7: Система всегда вернется в свое начальное состояние

$AGEFinit$ в CTL (в PLTL формулы для выражения этого нет)

Пример: контроллер светофора

Спецификация:

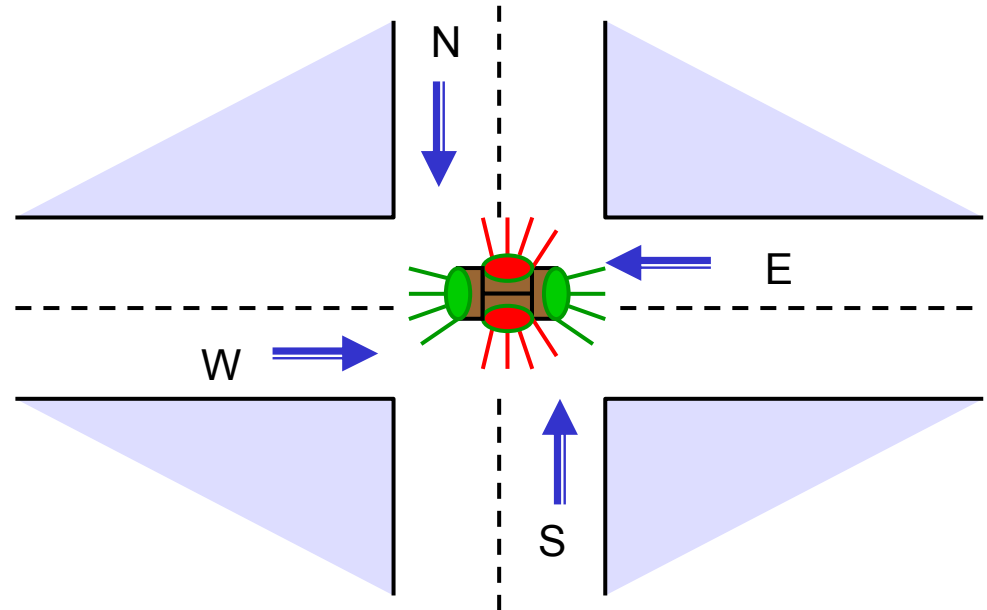
Построить контроллер, обеспечивающий безопасный и эффективный проезд перекрестка

Требования (неформально):

C1: Отсутствие коллизий

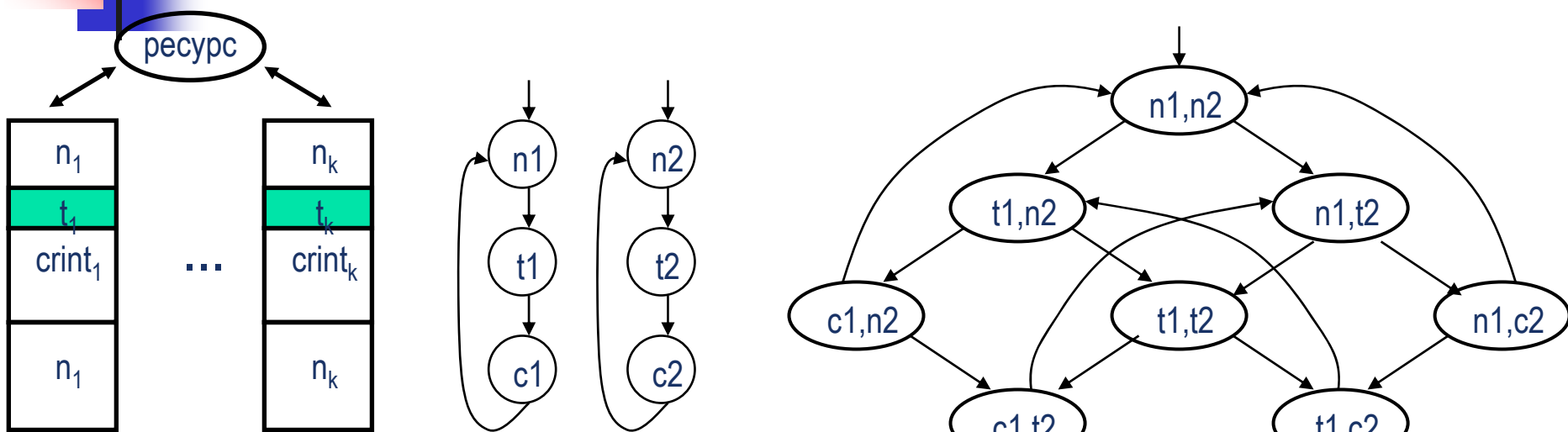
C2: Обеспечение проезда (прогресс)

Если нет машины, то переключения светофора может не быть



- C1: Безопасность
 - $AG \neg (E_green \wedge (N_green \vee S_green))$
 - $AG \neg (N_green \wedge (E_green \vee W_green))$
 - ...
- C2: Живость $AG (req \Rightarrow AF ack)$
 - $AG(N_ready \Rightarrow AF N_green)$
 - $AG(E_ready \Rightarrow AF E_green)$
 - ...

Пример: протокол взаимного исключения



Цель протокола – допустимый интерливинг

Safety – Не более одного процесса может находиться в критическом интервале
 $AG \neg(c1 \wedge c2)$ (но и протокол, вовсе не допускающий в *crint*, будет safe)

Liveness – Если какой-нибудь процесс запросит ресурс, ему в конце концов будет это разрешено $AG(t1 \Rightarrow AFc1)$
 Не выполняется!

Deadlock freeness – процесс всегда может запросить ресурс (вход в *crint*)
 $AG(n1 \Rightarrow EXt1)$

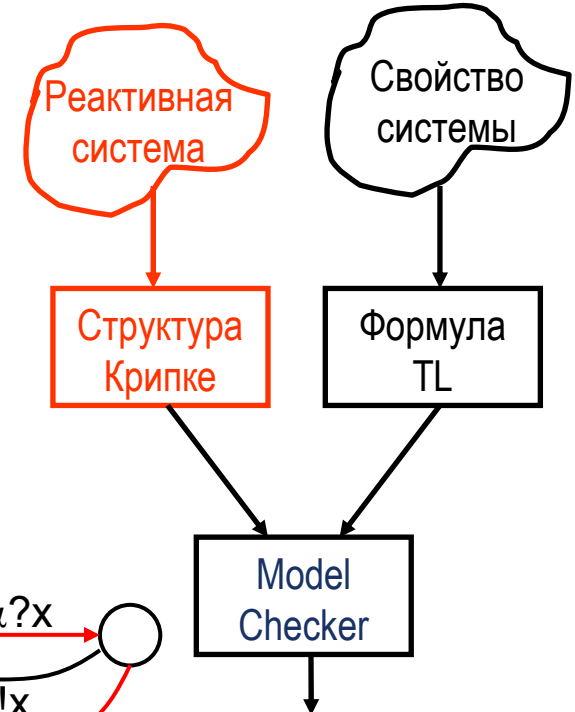
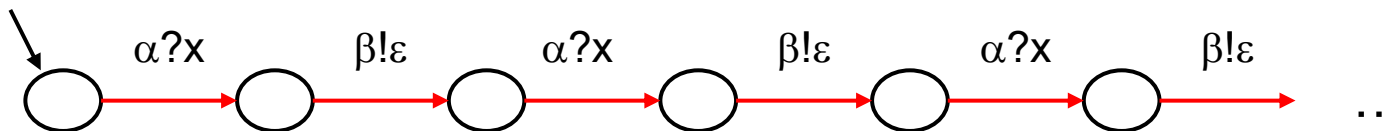
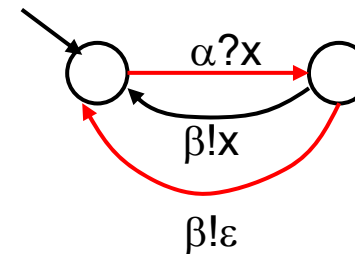
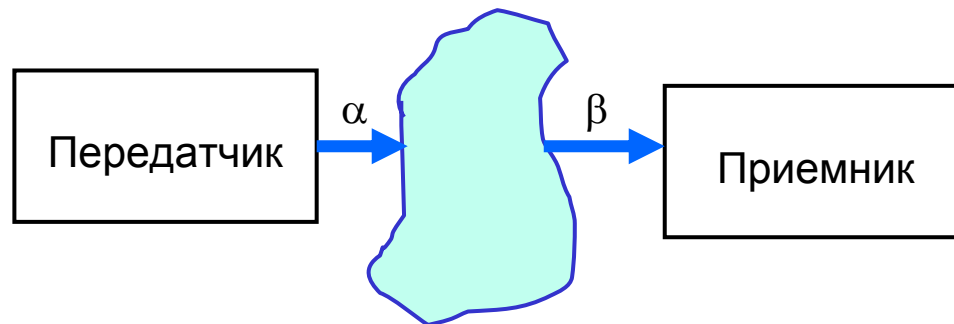
No strict sequencing – отсутствие одной последовательности разрешений
 $EF(c1 \wedge E[c1U(\neg c1 \wedge E[\neg c2Uc1])])$

Справедливость (fairness)

Достижимость
Безопасность
Дедлоки
Живость
Справедливость

Формальная модель строится для реальной системы, и в ней могут быть такие вычисления, которые не случаются в реальной системе

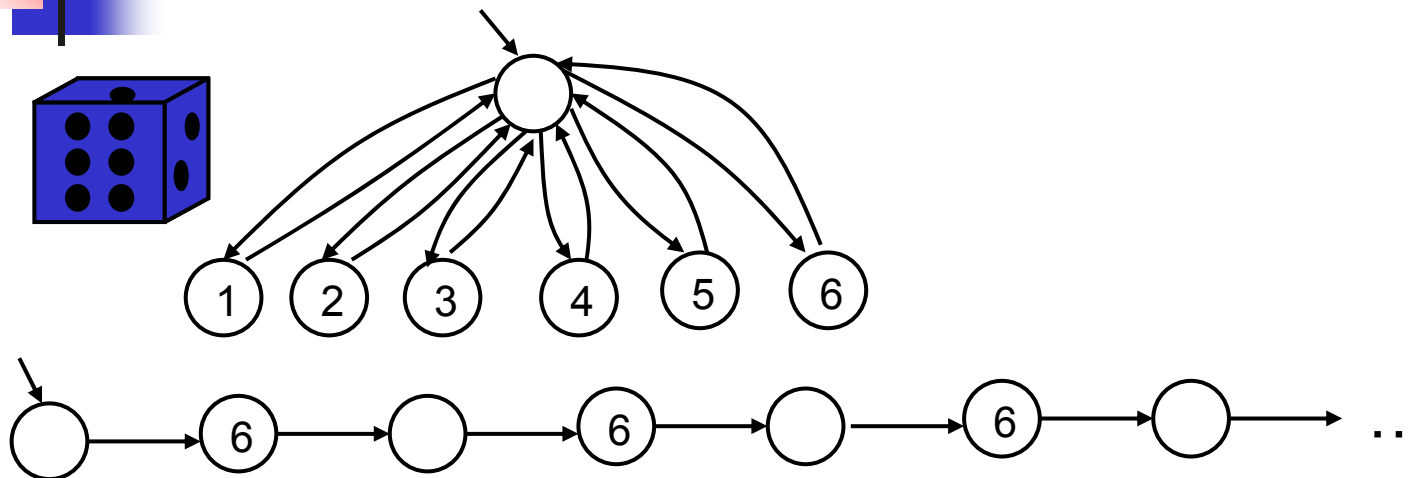
Например: В модели канала с потерями не может быть вычислений, в которых бесконечно долго не проходит сообщение (мы абстрагируемся от конечного значения вероятности потерь)



В модели структуры Крипке не можем выразить, что цепочка, состоящая только из потерь, невозможна (она имеет нулевую вероятность)

Модель бросания игральной кости

Достижимость
Безопасность
Дедлоки
Живость
Справедливость



В модели возможны вычисления, на которых некоторые грани никогда не выпадают. Это нереалистичные траектории. Вычисления, которые могут выполняться на практике, это такие, в которых каждая грань выпадает неопределенно часто – именно это выполняется в реальности.

(мы абстрагируемся от значений вероятности выпадения граней)

В модели структуры Крипке не можем выразить, что цепочка, состоящая только из потерь, невозможна (она имеет нулевую вероятность)

Нереалистичные траектории называются несправедливыми (unfair)



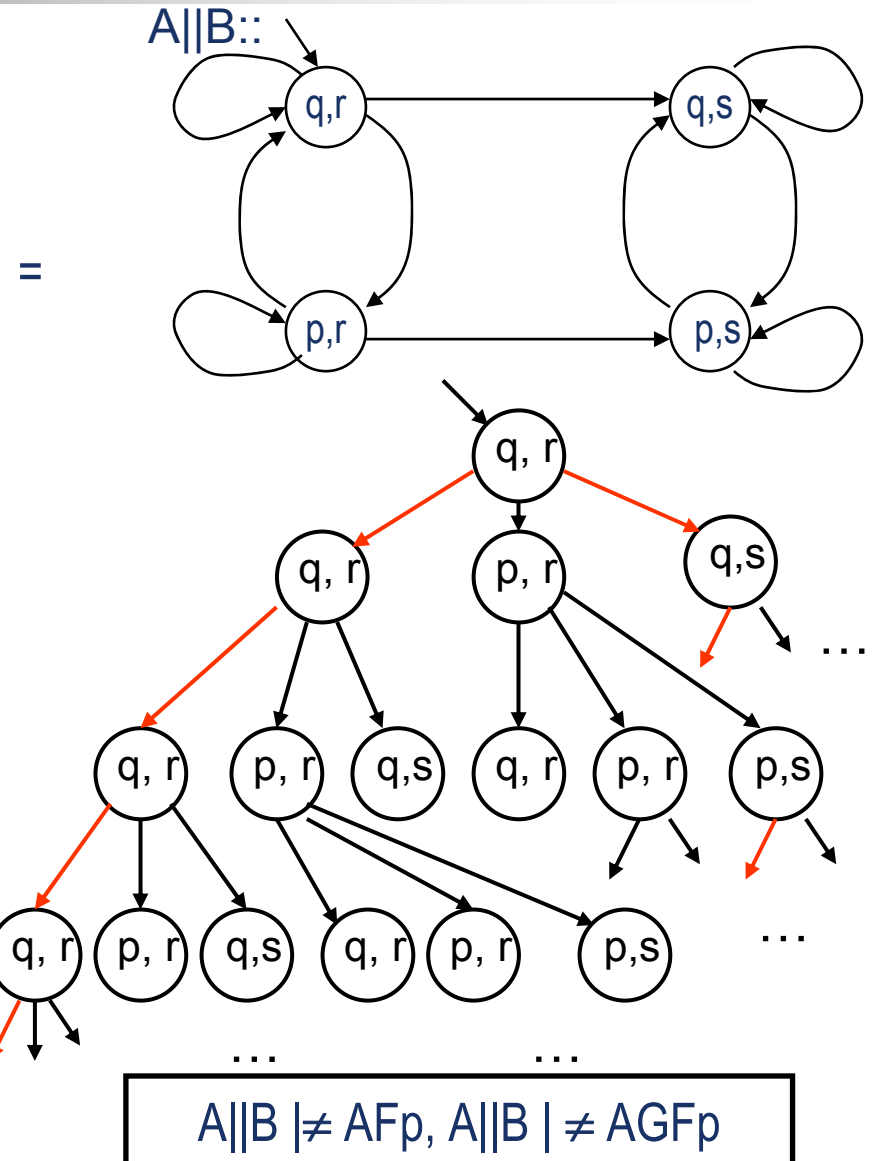
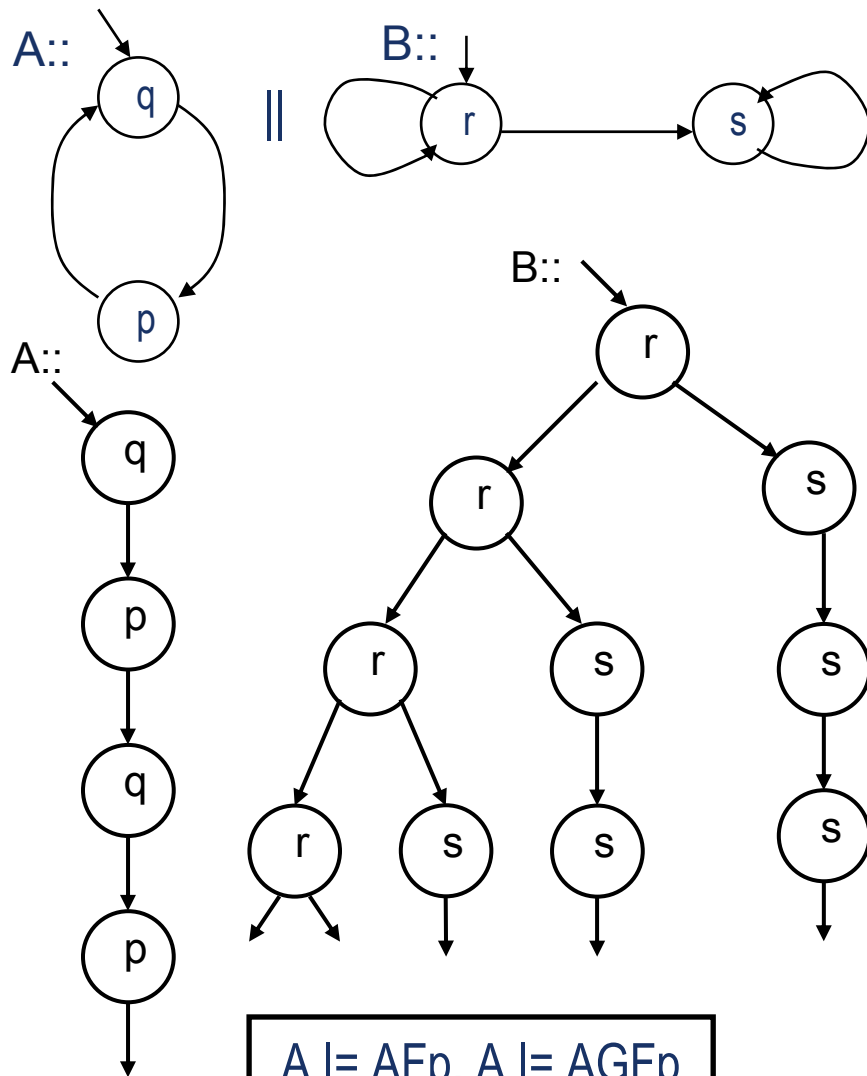
Примеры несправедливых вычислений

Достижимость
Безопасность
Дедлоки
Живость
Справедливость

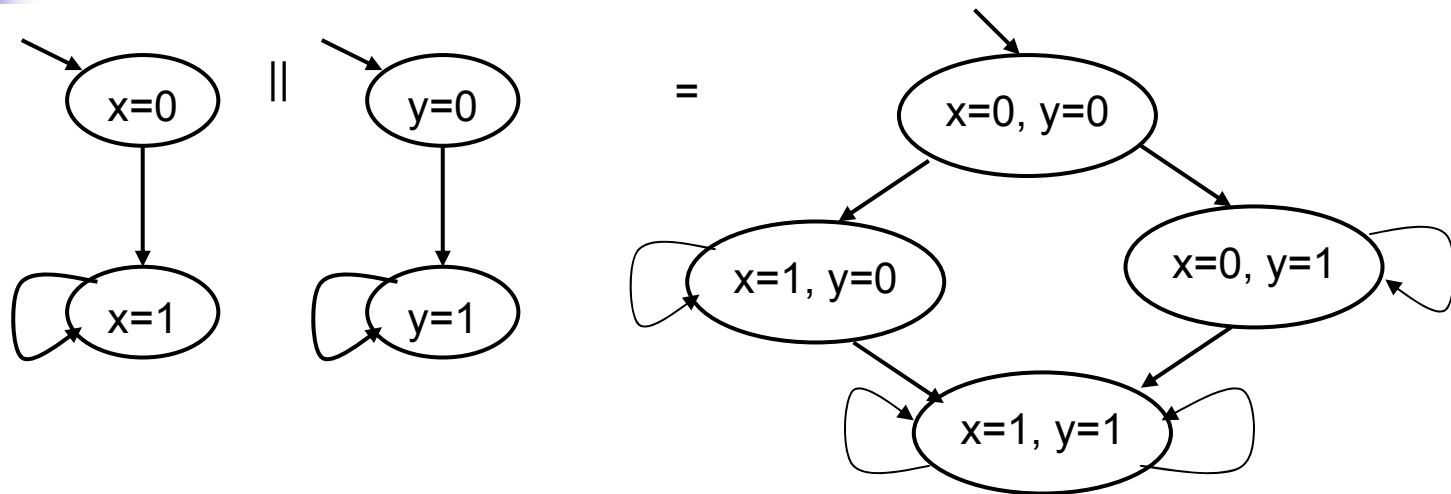
Модель чередования функционирования нескольких асинхронных параллельных процессов (interleaving): в каждом глобальном состоянии системы возможен шаг ЛЮБОГО готового процесса. Несправедливым (бесконечным) вычислением является такое, в котором один из процессов не делает ни одного шага (или делает только КОНЕЧНОЕ число шагов)

-(мы абстрагируемся от конечного значения времени выполнения операций)

Интерливинг в || композиции процессов



Пример с параллельными процессами



В глобальном состоянии $\langle x=1, y=0 \rangle$ возможны выполнения операций как первого, так и второго процессов. На вычислении, в котором выполняется операция только первого процесса, система заводится в состоянии $\langle x=1, y=0 \rangle$.

Такая траектория вычислений несправедлива!



Примеры несправедливых вычислений

Модель с критическим интервалом: один из процессов бесконечное число шагов находится в своем критическом интервале, что ограничивает доступ других процессов

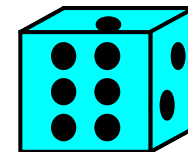
- (мы абстрагируемся от времени выполнения операций)

Модель выполнения процессов: то вычисление, в котором в альтернативном выборе одна из альтернатив бесконечно не выбирается, является нереальным (несправедливым)

Модель доступа нескольких параллельных процессов к ресурсу: один процесс бесконечно не получает ресурс



Как бороться?



Остаемся в рамках той же модели, но вводим ограничение: те вычисления, которые являются реалистичными, назовем “Справедливыми” (FAIR), и попытаемся при анализе рассматривать только их

Например, для игральной кости. Требование справедливости:

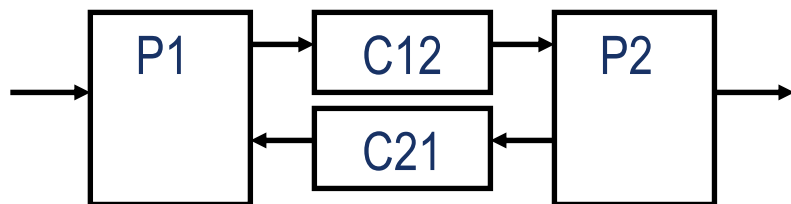
$$\Phi = A(GF1 \wedge GF2 \wedge GF3 \wedge GF4 \wedge GF5 \wedge GF6)$$

Требование Φ : на вычислении каждая грань выпадает неопределенно часто — именно это выполняется в реальности

Это ДОПОЛНИТЕЛЬНОЕ требование к тем свойствам, которые будем проверять. Только если на вычислении выполняется это свойство, мы будем это вычисление анализировать на предмет выполнения свойств верификации

Введение условия (требования) справедливости дает возможность отбросить НЕРЕАЛИСТИЧНЫЕ цепочки, являющиеся результатом использования упрощенной модели

Пример формулировки свойств: АВ протокол



Любое полученное на выходе сообщение было послано передатчиком – safety

Цепочка сообщений на выходе = начальный отрезок посланных - safety

Любое посланное сообщение будет в конце концов получено

$AG(\text{emitted} \Rightarrow F \text{ received})$ – liveness

Это свойство не выполняется, если модель допускает потери в канале

Fairness здесь – это предположение об отсутствии постоянных потерь:

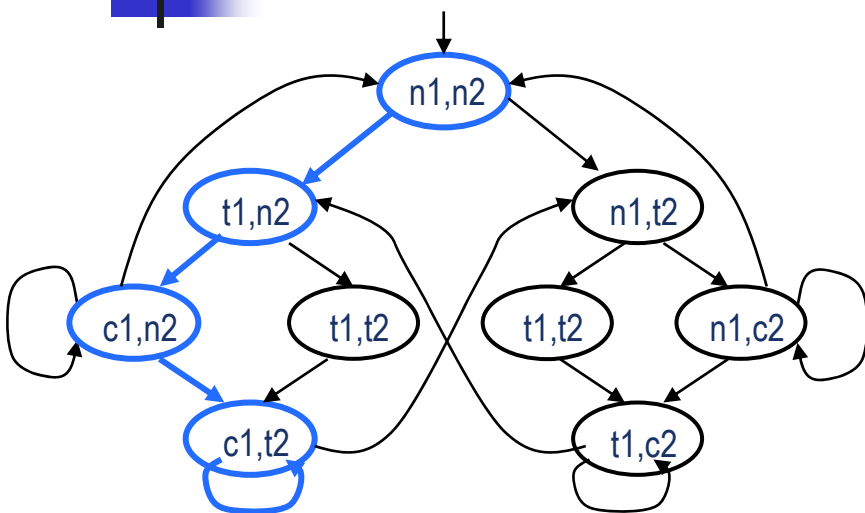
$A(GF \neg \text{loss} \Rightarrow G(\text{emitted} \Rightarrow F \text{ received}))$

Это можно использовать в качестве гипотезы (ограничения) справедливости

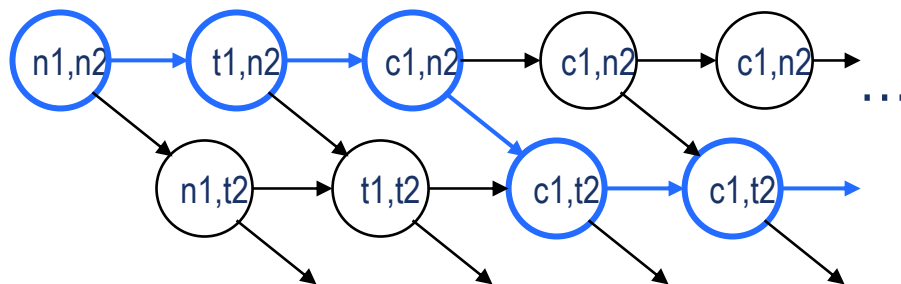
Некоторые верификаторы (SMV) рассматривают модель как пару:

< структура Крипке; гипотезы справедливости >

Model checking with Fairness



Модель упрощена: что, если шаги процессов – не на каждый тик, а возможны циклы у каждого состояния?



Liveness: $AG(t2 \Rightarrow AFc2)$ – не выполняется!

$M, s_0 \models \Phi$ не выполняется из-за нереалистичных поведений в модели M

1 путь: Изменить модель;

2 путь: Использовать фильтр при проверке свойств : $M, s_0 \models (\phi \Rightarrow \Phi)$

3 путь: **Fairness constrains** – предположение, что ψ_i истинны неопределенно часто на вычислениях, и проверять **только такие пути**: **fair** computational paths – E_f и A_f – на **fair** путях, удовлетворяющих сформулированным ограничениям $\{\psi_1, \psi_2, \dots, \psi_n\}$.

Свойства и гипотезы справедливости

Процесс может оставаться в состоянии (критической секции) произвольно долго, но когда-то выйдет из нее:

неверно, что всегда будет выполняться p : $\neg FGp$

но $\neg AFGp = E \neg FGp = EGF \neg p$ –
это значит, что неопределенно часто будет
выполняться $\neg p$

Требование (гипотеза) справедливости: GFq ,
т.е. неопределенно часто будет выполняться q

GFq – простое требование справедливости

$GFrequest_q \Rightarrow GFq$ – сильное требование
справедливости



Примеры требований справедливости

F1: *Если доступ в критическую секцию запрашивается неопределенно часто, то доступ будет разрешен тоже неопределенно часто:*

$A(GF \text{ crit_req} \Rightarrow GF \text{ crit_in})$

F2: *Если запрос посылается неопределенно часто, то подтверждение будет дано тоже неопределенно часто:*

$A(GF \text{ request} \Rightarrow GF \text{ acknowledge})$

GF выражает свойство “неопределенно много раз”

В CTL справедливость выразить нельзя. $AGFp$ не является формулой CTL

Хотя $AGFp \equiv AGAFp$, для формул $EGFp$, $A(GFp \wedge GFq)$, $EFGp$ и т.д. эквивалентных в CTL нет.

Можно, однако, требование справедливости удовлетворить по другому



Как ввести требование справедливости?

Один из возможных методов ограничить в модели те пути, которых не может быть в реальной системе – ввести требование “справедливости” путей (“fairness”)

Зададим конечный список F подмножеств состояний Q структуры Крипке вида: $\{F_i \mid i=1, \dots, k\}$, $F_i \subseteq Q$;

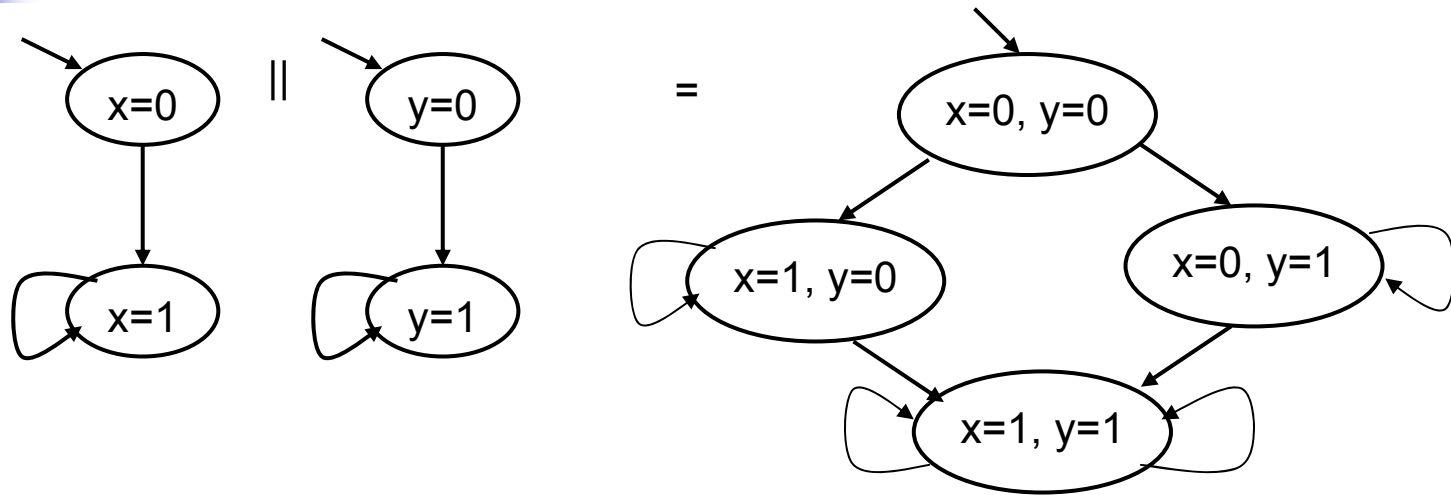
Будем считать, что “справедливое” вычисление π - это такое вычисление, которое посещает каждое из множеств состояний F_i бесконечное число раз (на справедливом пути хотя бы один элемент из каждого ограничения справедливости должен встречаться бесконечно много раз)

Поэтому условие *fairness* вычисления можно формализовать так:

Путь π *справедливый* iff $\forall i \in \{1, \dots, k\} \quad \text{inf}(\pi) \cap F_i \neq \emptyset$

Множества состояний F_i можно описать при помощи формул пропозициональной логики. Пусть каждое из множеств F_i задано формулами $\phi_1, \phi_2, \dots, \phi_k$ над атомными предикатами структуры Крипке

Пример с параллельными процессами



На вычислении, в котором выполняется операция только первого процесса, система зависает в состоянии $\langle x=1, y=0 \rangle$.

На вычислении, в котором выполняется операция только второго процесса, система зависает в состоянии $\langle x=1, y=0 \rangle$.

Такие траектории вычислений несправедливы!

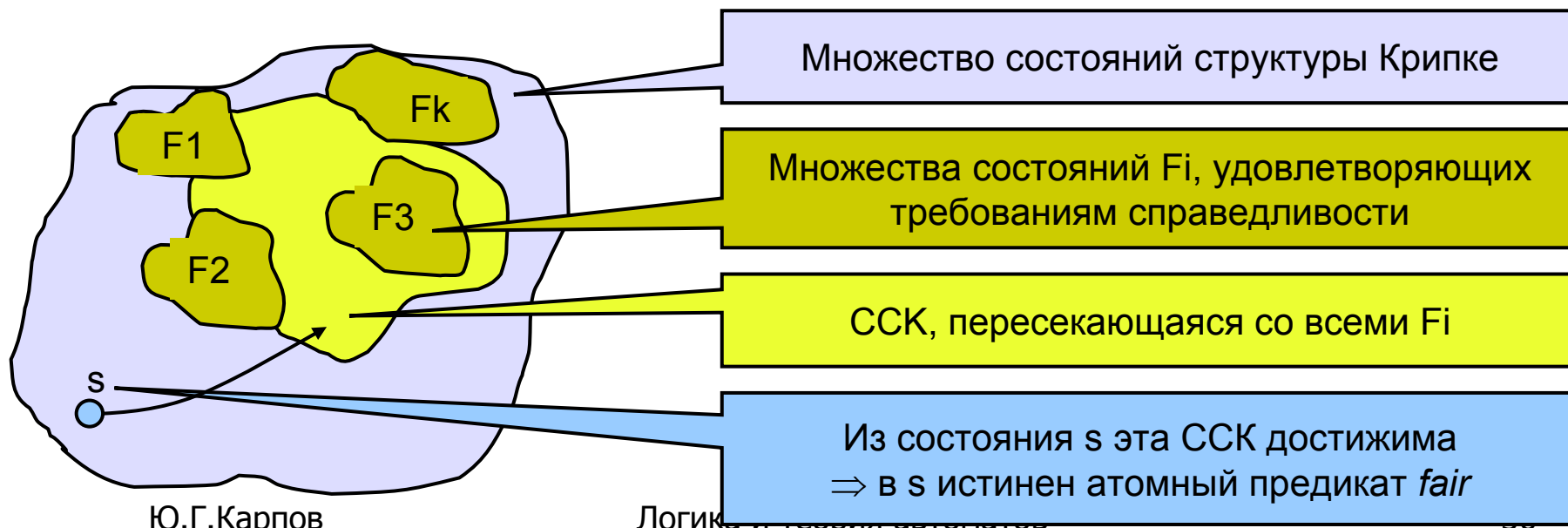
Условие справедливости для данного примера формулируем как одно множество состояний $F1 = \{\langle x=1, y=1 \rangle\}$. Состояние $\langle x=1, y=1 \rangle$ должно неопределенно часто встретиться в каждой траектории модели, на которой мы хотим проверять СВОЙСТВА СИСТЕМЫ

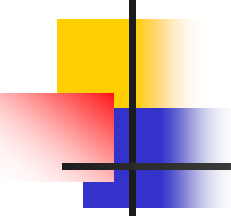
Формализация требований справедливости

Пусть множество пропозициональных формул $\phi_1, \phi_2, \dots, \phi_k$ определяет требование справедливости. Как при верификации проверять только те пути, на которых все ϕ_i становятся истинными бесконечное число раз?

Обозначим F_i множество состояний, удовлетворяющих формуле ϕ_i .

Введем атомный предикат *fair* – ему удовлетворяют все состояния, из которых *существует* вычисление, проходящее через каждое из множеств F_1, F_2, \dots, F_k бесконечное число раз. Тогда в состоянии s предикат *fair* истинен *тогда*, когда в структуре Крипке существует достижимая из s ССК, которая пересекается со всеми множествами F_1, F_2, \dots, F_k (фактически, выполняется $s \models_{\text{fair}} \text{EG true}$)





Проверка свойств CTL с требованием справедливости (базис EX, EU, EG)

Верификацию с дополнительным условием справедливости можно выполнить так:

1. Находим все состояния, в которых удовлетворяется предикат *fair*
2. По подформулам формулы ϕ вычисляем $s_{\text{fair}}(\phi)$ так:

$$s_{\text{fair}}(\phi) \equiv s \models (\text{fair} \wedge \phi)$$

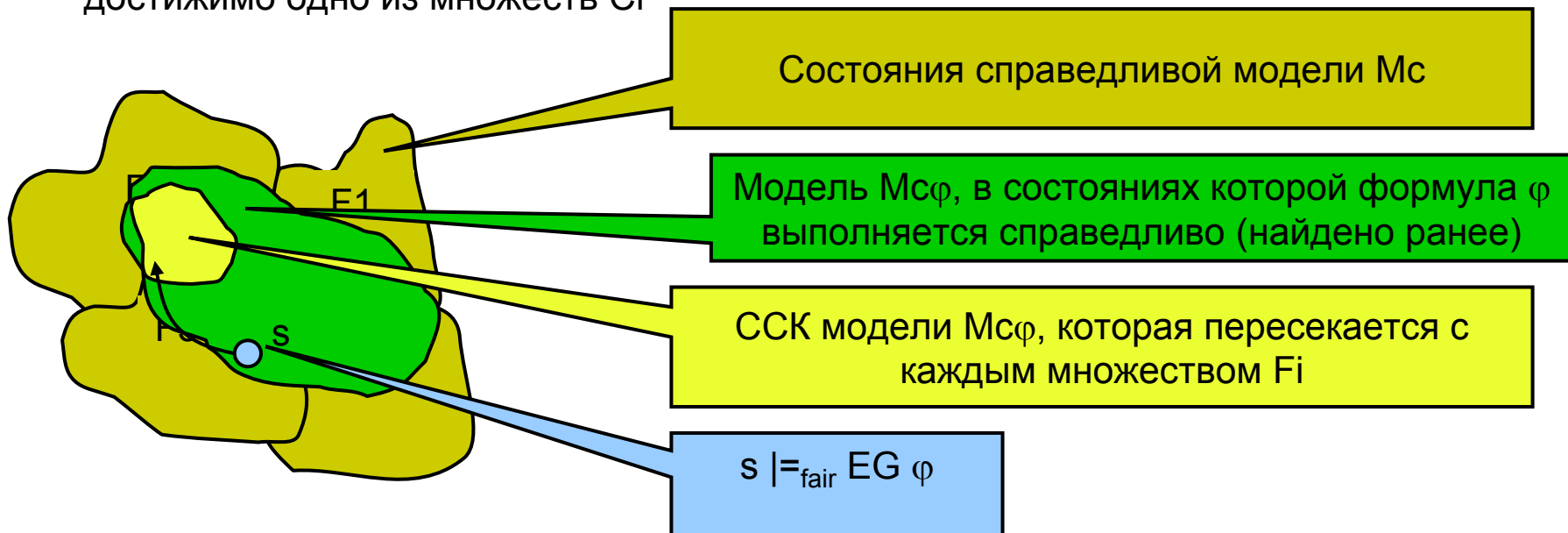
$$s_{\text{fair}}(\text{EX}\phi) \equiv s \models (\text{EX}(\text{fair} \wedge \phi))$$

$$s_{\text{fair}}(\text{E}(\psi \cup \eta)) \equiv s \models (\text{E}(\psi \cup (\text{fair} \wedge \eta)))$$

3. Вычисление $s \models_{\text{fair}} \text{EG}\psi$ требует чуть большего внимания

Проверка подформулы $EG\varphi$ с требованием справедливости

1. Ограничиваем структуру Крипке до такой модели M_s , в которой нет состояний, не относящихся к множествам F_1, \dots, F_k (так называемую “справедливую” модель)
2. Ограничиваем *справедливую* модель такой $M_{s\varphi}$, во всех состояниях которой формула φ выполняется справедливо (все такие состояния найдены на предыдущих шагах, поскольку мы проводим верификацию индукцией по подформулам формулы φ)
3. Находим $\{C_1, C_2, \dots, C_m\}$ – множество ССК, которые пересекаются с КАЖДЫМ F_i .
4. Формула $EG\varphi$ *справедливо* выполняется в состоянии s *тогда*, когда из s достижимо одно из множеств C_i



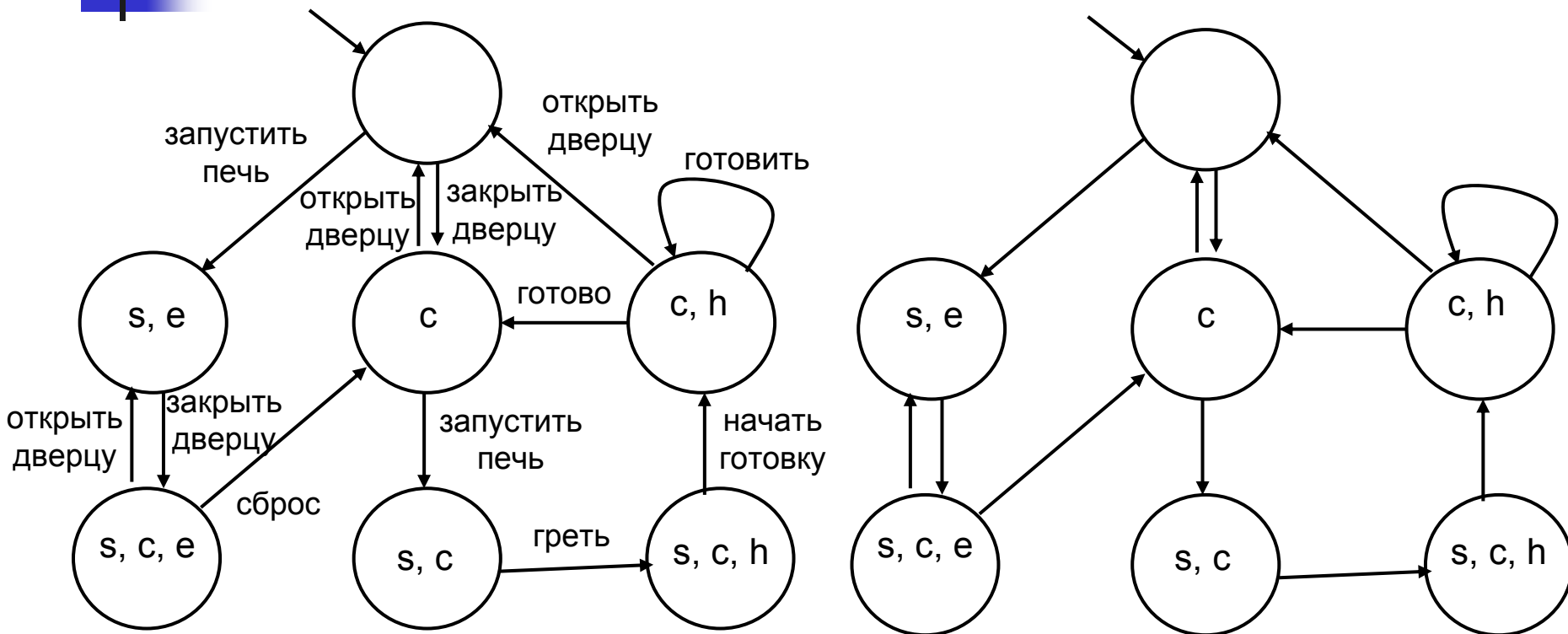


Fair Model Checking: CTL и LTL

CTL: Для каждой формулы ϕ CTL определяем *fair*-значение этой формулы. В каждом состоянии q можем определить, выполняется или не выполняется формула ϕ с дополнительным условием справедливости.

LTL: Обобщенный автомат Бюхи, построенный по структуре Крипке для исходной системы, изменяется – для поиска контрпримера в качестве заключительных подмножеств состояний он будет иметь дополнение до множеств F_i , т.е. $2^Q \setminus \{F_i\}$

Пример: Микроволновая печь



Атомарные предикаты:

Start – s

Close – c

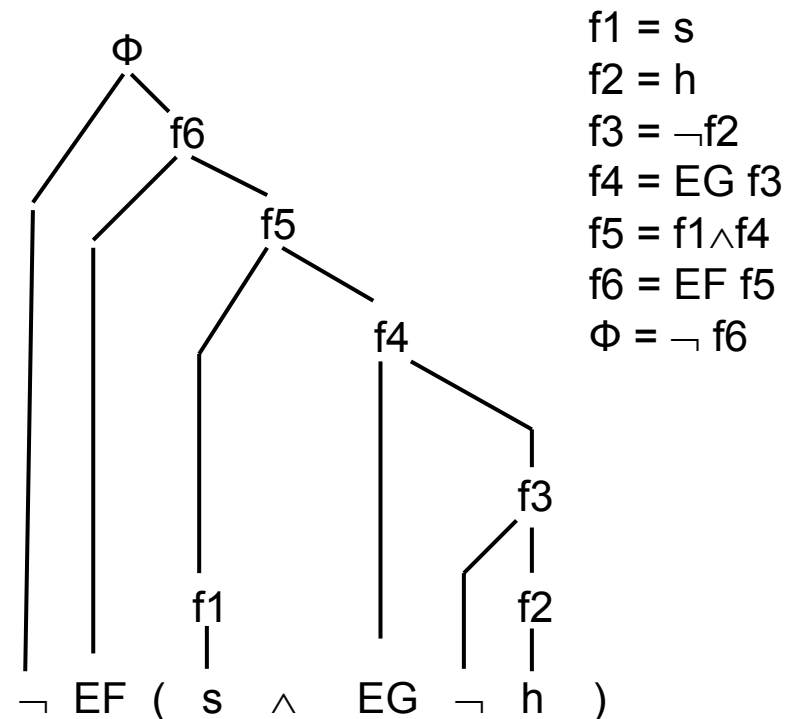
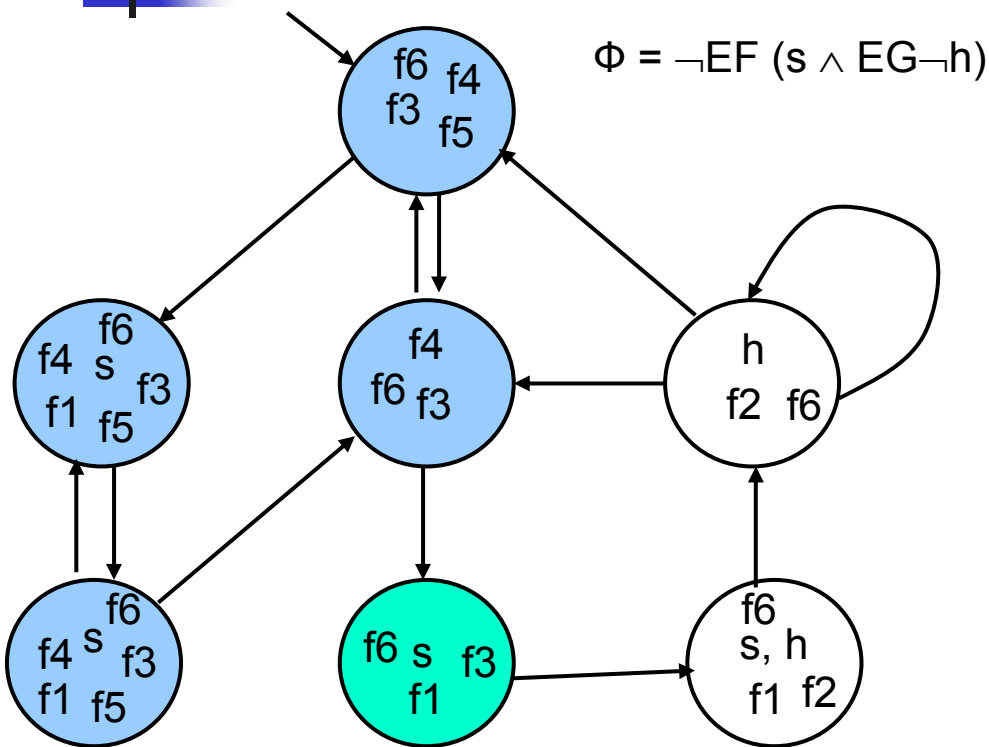
Heat - h

Error - e

$$\Phi = AG (Start \Rightarrow AF Heat) = \neg EF (Start \wedge EG \neg Heat)$$

$$\Phi = \neg EF (s \wedge EG \neg h)$$

Построение подформул формулы Φ



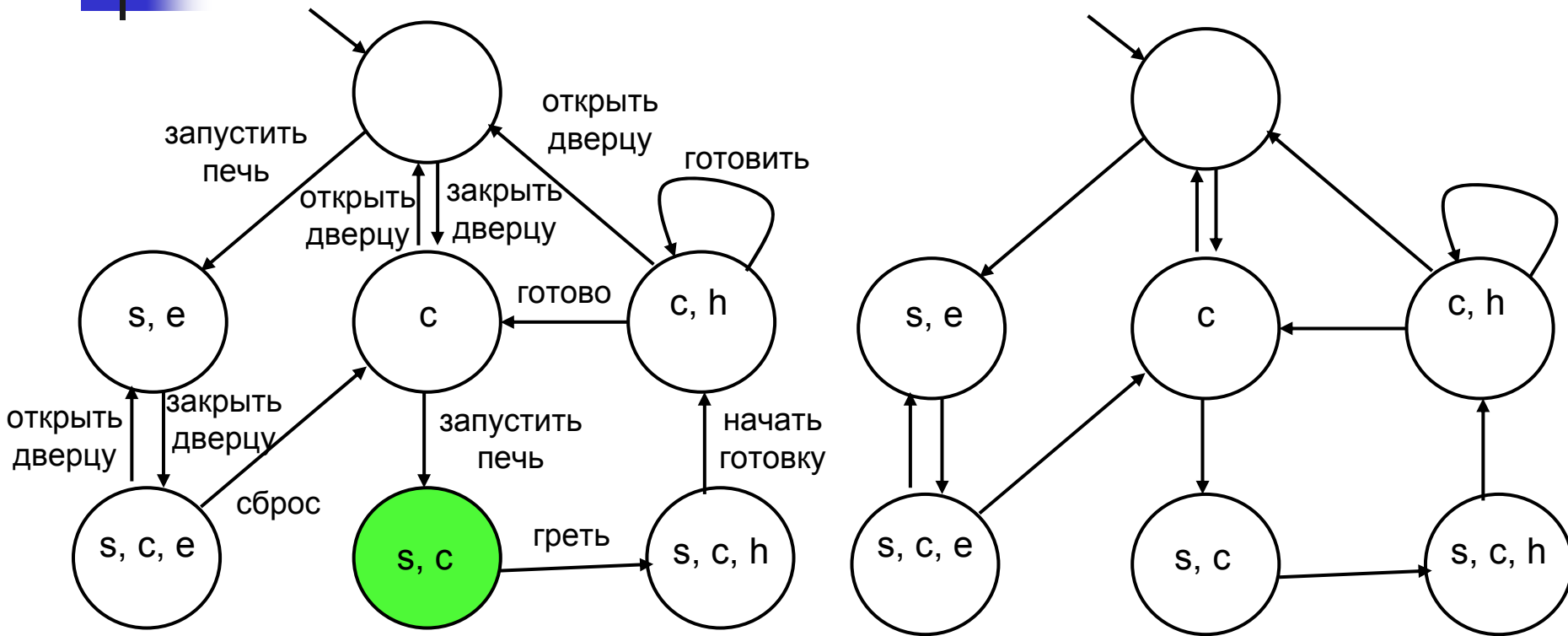
Подструктура Крипке, удовлетворяющая $f3$, выделена

Ее ССК выделена голубым цветом

Формула $f6$ истинна во всех состояниях

Формула Φ не истинна ни в одном состоянии

Анализ с учетом справедливости



Атомные предикаты:

Start – s
Close – c
Heat – h
Error – e

Ограничение справедливости:

Пользователь правильно работает с печью, если
вычисление проходит через $\text{Start} \wedge \text{Close} \wedge \neg \text{Error}$

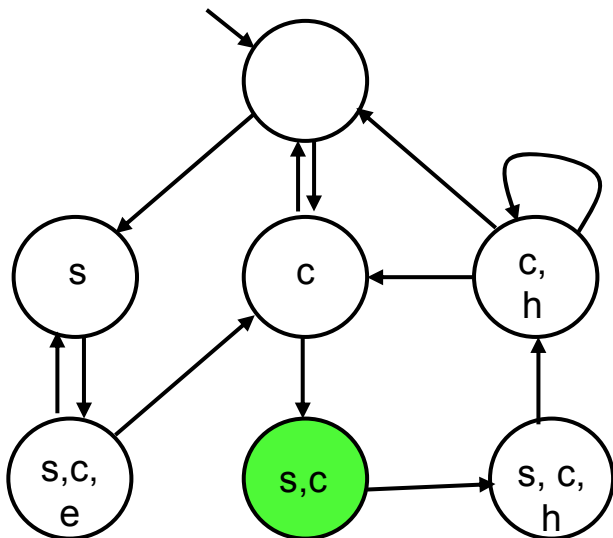
$$\Phi = \text{AG} (\text{Start} \Rightarrow \text{AF Heat}) = \neg \text{EF} (\text{Start} \wedge \text{EG} \neg \text{Heat})$$

$$\Phi = \neg \text{EF} (s \wedge \text{EG} \neg h)$$

Вычисление справедливой $\Phi = \neg EF (s \wedge EG \neg h)$

1. Находим все состояния, в которых удовлетворяется предикат *fair* (*у нас – все*)
2. По подформулам ϕ вычисляем $s_{\text{fair}}(\phi)$ так: (*у нас – не изменяются*)

$s_{\text{fair}}(\phi) \equiv s \models (\text{fair} \wedge \phi)$	$f1 = s$
$s_{\text{fair}}(EX\phi) \equiv s \models (EX (\text{fair} \wedge \phi))$	$f2 = h$
$s_{\text{fair}}(E(\psi \cup \eta)) \equiv s \models (E(\psi \cup (\text{fair} \wedge \eta)))$	$f3 = \neg f2$
	$f4 = EG f3$
	$f5 = f1 \wedge f4$
	$f6 = EF f5$
	$\Phi = \neg f6$
3. Вычисление $s \models_{\text{fair}} EG f$ состоит в том, что выделяются ССК, помеченные f , которые пересекаются с каждым $\{Fi\}$ и эти ССК и путь в них удовлетворяют $\text{fair } EG f$



В состоянии s предикат *fair* истинен *тогда*, когда на структуре Крипке существует такая ССК, достижимая из s , которая пересекается со всеми множествами $F1, F2, \dots, Fk$. У нас одно Fi с одним состоянием, из всех состояний оно достигается, поэтому *fair* истинен во всех состояниях

У нас – $f4=f5=f6$ =пустые множества, а формула Φ – истинна везде



Заключение

Выбор множества проверяемых формул осуществляется человеком – верификатором, никто не может сказать, насколько ПОЛНО осуществлена проверка – все ли нужные свойства системы проверены

Определение классов свойств позволяет не пропустить при верификации некоторые важные типы свойств

Достижимость:

некоторое конкретное состояние будет достигнуто (reachability)

Безопасность:

ничто плохое не произойдет (safety)

Живость:

ничто хорошее выполнится (liveness)

Свобода от дедлоков (deadlock freeness, non blocking)

процесс не будет заблокирован другим процессом

Справедливость (fairness):

ограничение нереальных или нежелательных траекторий

- Выразить в LTL свойства:
 - 1. Если произойдет p , то в будущем q никогда не случится
 - 2. Если произойдет p , то когда-нибудь в будущем выполнится q , а сразу после этого произойдет r
 - 3. Если случится p , то в будущем случится q , а между ними не будет выполнено r
 - 4. В будущем p может случиться не более, чем один раз
 - 5. В будущем p случится ровно один раз
 - 6. Построить структуру Крипке, удовлетворяющую формулам, построенным по заданиям 1-5



Спасибо за внимание