Методы верификации программ и схем

ЛЕКТОРЫ:

Владимир Анатольевич Захаров
Владислав Васильевич Подымов

zakh@cs.msu.su

Лекция 5.

Верификация моделей программ

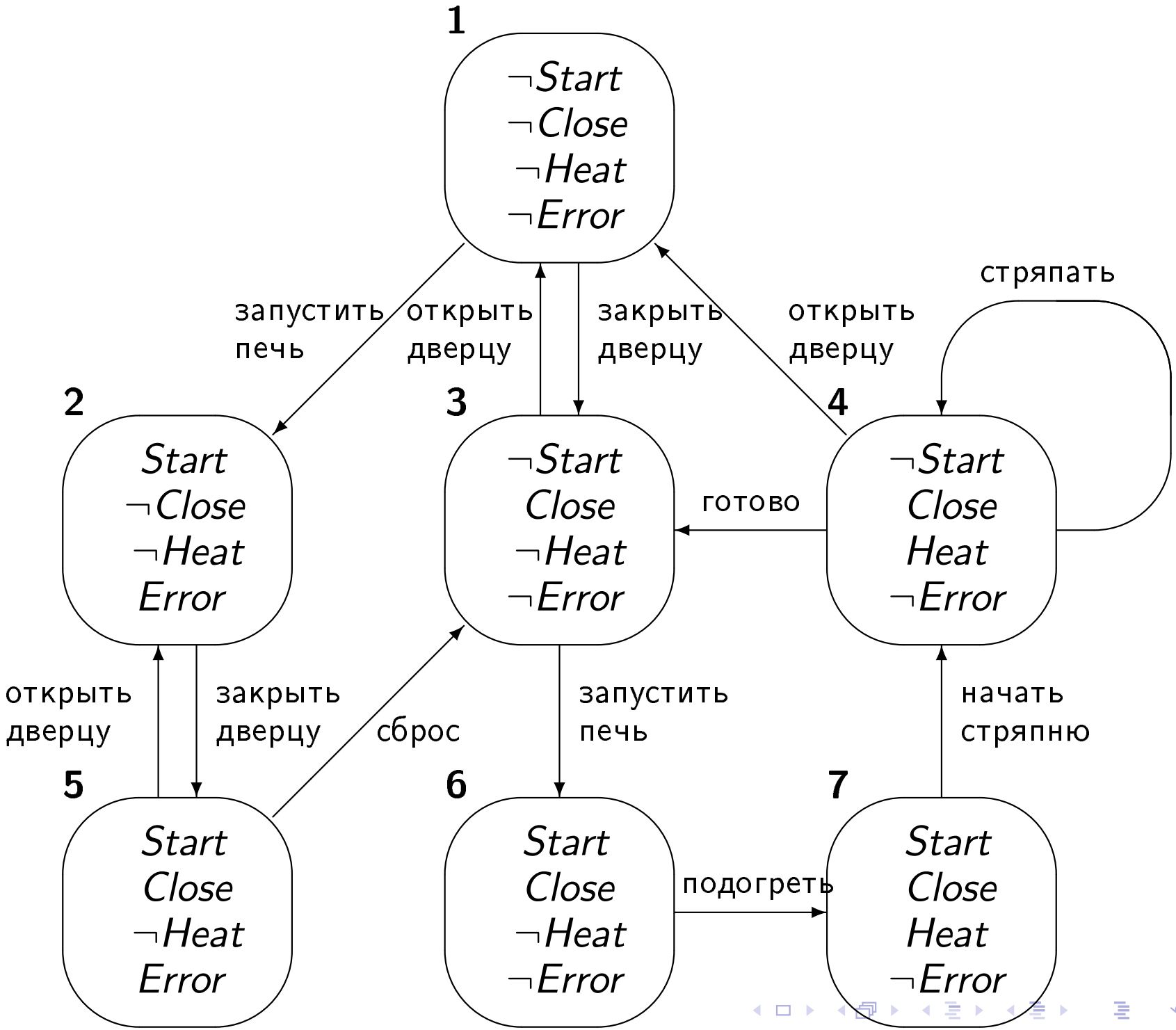
1. Задача model checking для CTL
2. Табличный алгоритм model checking для CTL
3. Model checking для CTL в ограничениях справедливости
4. Двоичные разрешающие диаграммы (BDDs)
5. Построение разрешающих диаграмм
6. Представление моделей Крипке двоичными разрешающими диаграммами

Задача model checking для CTL

Рассмотрим множество атомарных высказываний AP.

Моделью Крипке M над множеством AP назовем четверку $M = (S, S_0, R, L)$, в которой:

1) S — конечное множество состояний;
2) $S_0 \subseteq S$ — множество начальных состояний;
3) $R \subseteq S \times S$ — отношение переходов, которое обязано быть тотальным, т. е. для каждого состояния $s \in S$ должно существовать такое состояние $s' \in S$, что имеет место $R(s, s')$;
4) $L\colon S \to 2^{AP}$ — функция разметки, которая помечает каждое состояние множеством атомарных высказываний, истинных в этом состоянии.


1
¬Start
¬Close
¬Heat
¬Error
запустить печь
открыть дверцу
закрыть дверцу
открыть дверцу
стряпать
2
Start
¬Close
¬Heat
Error
3
¬Start
Close
¬Heat
¬Error
готово
4
¬Start
Close
Heat
¬Error
открыть дверцу
закрыть дверцу
сброс
запустить печь
начать стряпню
5
Start
Close
¬Heat
Error
6
Start
Close
¬Heat
¬Error
подогреть
7
Start
Close
Heat
¬Error

Задача model checking для CTL

Формулы логики CTL строятся из атомарных высказываний (базовых свойств) AP при помощи булевых связок $\neg, \wedge, \vee, \rightarrow, \ldots$ и десяти основных операторов CTL:

- **AX** и **EX**,
- **AF** и **EF**,
- **AG** и **EG**,
- **AU** и **EU**,
- **AR** и **ER**.

Задача model checking для CTL

Примеры CTL спецификаций:

- $\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat})$,

Задача model checking для CTL

Примеры CTL спецификаций:

- $\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat})$,
- $\mathbf{AG}(\mathit{Error} \to \mathbf{A}[\neg \mathit{Start}\, \mathbf{R}\, \mathit{Error}])$,

Задача model checking для CTL

Примеры CTL спецификаций:

- $\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat})$,
- $\mathbf{AG}(\mathit{Error} \to \mathbf{A}[\neg \mathit{Start}\, \mathbf{R}\, \mathit{Error}])$,
- $\mathbf{AG}\, \mathbf{EX}\, \mathbf{EX}\, \mathbf{EX}\, \mathit{Heat}$,

Задача model checking для CTL

Примеры CTL спецификаций:

- $\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat})$,
- $\mathbf{AG}(\mathit{Error} \to \mathbf{A}[\neg \mathit{Start}\, \mathbf{R}\, \mathit{Error}])$,
- $\mathbf{AG}\, \mathbf{EX}\, \mathbf{EX}\, \mathbf{EX}\, \mathit{Heat}$,
- $\neg\, \mathbf{EG}(\mathit{Error} \to \mathbf{AX}\, \mathit{Error})$,

Задача model checking для CTL

Примеры CTL спецификаций:

- $\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat})$,
- $\mathbf{AG}(\mathit{Error} \to \mathbf{A}[\neg \mathit{Start}\, \mathbf{R}\, \mathit{Error}])$,
- $\mathbf{AG}\, \mathbf{EX}\, \mathbf{EX}\, \mathbf{EX}\, \mathit{Heat}$,
- $\neg\, \mathbf{EG}(\mathit{Error} \to \mathbf{AX}\, \mathit{Error})$,
- $\mathbf{AG}(\mathbf{A}[\neg \mathit{Start}\, \mathbf{U}\, \mathit{Close}])$.

Задача model checking для CTL

Пусть задана модель Крипке $M = (S, S_0, R, L)$, представляющая информационную систему с конечным множеством состояний S, и формула темпоральной логики φ, которая выражает некоторую желаемую спецификацию. Требуется найти в множестве S подмножество S_φ всех состояний s, в которых выполняется φ, т.е. множество

$$S_\varphi = \{s \in S \mid M, s \models \varphi\},$$

и проверить выполнимость включения $S_0 \subseteq S_\varphi$.

Задача model checking для CTL

Первые алгоритмы решения задачи верификации моделей использовали **явное** представление моделей Крипке в виде размеченных ориентированных графов, дуги которых задавали при помощи ссылок.

В этом случае вершины представляют состояния из S , дуги графа соответствуют отношению переходов R , а разметка вершин определяется функцией $L: \ S \to 2^{AP}$.

Алгоритмы верификации моделей, работающие с явным представлением моделей, получили название **табличных алгоритмов** , поскольку они итеративно заполняли таблицу выполнимости подформул заданной формулы CTL во всех состояниях модели.

Табличный алгоритм model checking для CTL

Применяя равносильные замены

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$;
- $f \to g \equiv \neg f \vee g$;
- $\mathbf{AX}\, f \equiv \neg\, \mathbf{EX}(\neg f)$;
- $\mathbf{EF}\, f \equiv \mathbf{E}[\mathit{True}\, \mathbf{U}\, f]$;
- $\mathbf{AG}\, f \equiv \neg\, \mathbf{EF}(\neg f)$;
- $\mathbf{AF}\, f \equiv \neg\, \mathbf{EG}(\neg f)$;
- $\mathbf{A}[f\, \mathbf{U}\, g] \equiv \neg\, \mathbf{E}[\neg g\, \mathbf{U}\, (\neg f \wedge \neg g)] \wedge \neg\, \mathbf{EG}\, \neg g$;
- $\mathbf{A}[f\, \mathbf{R}\, g] \equiv \neg\, \mathbf{E}[\neg f\, \mathbf{U}\, \neg g]$;
- $\mathbf{E}[f\, \mathbf{R}\, g] \equiv \neg\, \mathbf{A}[\neg f\, \mathbf{U}\, \neg g]$,

всякую CTL-формулу можно записать при помощи связок и операторов $\neg$, $\vee$, $\mathbf{EX}$, $\mathbf{EG}$ и $\mathbf{EU}$. Таким образом, достаточно уметь анализировать формулы шести типов в зависимости от того, является ли формула g атомарной или она представлена в одной из форм: $\neg f_1$, $f_1 \vee f_2$, $\mathbf{EX}\, f_1$, $\mathbf{EG}\, f_1$ и $\mathbf{E}[f_1\, \mathbf{U}\, f_2]$.

Табличный алгоритм model checking для CTL

$\mathbf{AG}(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$

Табличный алгоритм model checking для CTL

$$\mathbf{AG}(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

$$\neg\,\mathbf{EF}\,\neg(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

Табличный алгоритм model checking для CTL

$$\mathbf{AG}(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

$$\neg\,\mathbf{EF}\,\neg(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

$$\neg\,\mathbf{E}[\mathit{True}\,\mathbf{U}\,\neg(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat})] \equiv$$

Табличный алгоритм model checking для CTL

$$\mathbf{AG}(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

$$\neg\,\mathbf{EF}\,\neg(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

$$\neg\,\mathbf{E}[\mathit{True}\,\mathbf{U}\,\neg(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat})] \equiv$$

$$\neg\,\mathbf{E}[\mathit{True}\,\mathbf{U}\,\neg(\neg\mathit{Start} \vee \mathbf{AF}\,\mathit{Heat})] \equiv$$

Табличный алгоритм model checking для CTL

$$\mathbf{AG}(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

$$\neg\,\mathbf{EF}\,\neg(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat}) \equiv$$

$$\neg\,\mathbf{E}[\mathit{True}\,\mathbf{U}\,\neg(\mathit{Start} \rightarrow \mathbf{AF}\,\mathit{Heat})] \equiv$$

$$\neg\,\mathbf{E}[\mathit{True}\,\mathbf{U}\,\neg(\neg\mathit{Start} \vee \mathbf{AF}\,\mathit{Heat})] \equiv$$

$$\neg\,\mathbf{E}[\mathit{True}\,\mathbf{U}\,\neg(\neg\mathit{Start} \vee \neg\,\mathbf{EG}\,\neg\mathit{Heat})]$$

Табличный алгоритм model checking для CTL

Пусть задана модель Крипке $M = (S, S_0, R, L)$, и мы хотим определить, в каких состояниях выполняется CTL-формула φ.

Табличный алгоритм model checking для CTL

Пусть задана модель Крипке $M = (S, S_0, R, L)$, и мы хотим определить, в каких состояниях выполняется CTL-формула φ.

Работа алгоритма заключается в приписывании каждому состоянию s множества $label(s)$ тех подформул формулы φ, которые выполняются в состоянии s.

Табличный алгоритм model checking для CTL

Пусть задана модель Крипке $M = (S, S_0, R, L)$, и мы хотим определить, в каких состояниях выполняется CTL-формула φ.

Работа алгоритма заключается в приписывании каждому состоянию s множества $label(s)$ тех подформул формулы φ, которые выполняются в состоянии s.

Вначале $label(s) = L(s)$.

Табличный алгоритм model checking для CTL

Пусть задана модель Крипке $M = (S, S_0, R, L)$, и мы хотим определить, в каких состояниях выполняется CTL-формула φ.

Работа алгоритма заключается в приписывании каждому состоянию s множества $label(s)$ тех подформул формулы φ, которые выполняются в состоянии s.

Вначале $label(s) = L(s)$.

Далее на каждом шаге i обрабатываются подформулы, в которых глубина вложенности CTL-операторов равна $i - 1$. После обработки подформулу добавляют к множеству $label(s)$ всех тех состояний, в которых она выполнима.

Табличный алгоритм model checking для CTL

Пусть задана модель Крипке $M = (S, S_0, R, L)$, и мы хотим определить, в каких состояниях выполняется CTL-формула φ.

Работа алгоритма заключается в приписывании каждому состоянию s множества $label(s)$ тех подформул формулы φ, которые выполняются в состоянии s.

Вначале $label(s) = L(s)$.

Далее на каждом шаге i обрабатываются подформулы, в которых глубина вложенности CTL-операторов равна $i - 1$. После обработки подформулу добавляют к множеству $label(s)$ всех тех состояний, в которых она выполнима.

По окончании работы алгоритма мы обнаружим, что $S_\varphi = \{s \ : \ \varphi \in label(s)\}$.

Табличный алгоритм model checking для CTL

Для формул вида $\neg f_1$ мы полагаем

$$\neg f_1 \in label(s) \iff f_1 \notin label(s).$$

Табличный алгоритм model checking для CTL

Для формул вида $\neg f_1$ мы полагаем

$$\neg f_1 \in label(s) \iff f_1 \notin label(s).$$

Для формул вида $f_1 \vee f_2$ мы полагаем

$$f_1 \vee f_2 \in label(s) \iff f_1 \in label(s) \text{ или } f_2 \in label(s).$$

Табличный алгоритм model checking для CTL

Для формул вида $\neg f_1$ мы полагаем

$$\neg f_1 \in label(s) \iff f_1 \notin label(s).$$

Для формул вида $f_1 \vee f_2$ мы полагаем

$$f_1 \vee f_2 \in label(s) \iff f_1 \in label(s) \text{ или } f_2 \in label(s).$$

Для формул вида $\mathbf{EX}\, f_1$ мы полагаем

$$\mathbf{EX}\, f_1 \in label(s) \iff \exists s' : f_1 \in label(s') \text{ и } (s, s') \in R.$$

Табличный алгоритм model checking для CTL

Чтобы проанализировать формулу вида $g = \mathbf{E}[f_1 \mathbf{U} f_2]$, мы вначале выделим все состояния s', у которых $f_2 \in label(s')$.

Далее будем отступать из них назад, используя отношение, обратное отношению переходов R, двигаясь только по тем состояниям s, у которых $f_1 \in label(s)$.

Все выделенные таким образом состояния s' и s пометим формулой g.

Для выполнения этой процедуры требуется время $O(|S| + |R|)$.

Табличный алгоритм model checking для CTL

procedure CheckEU(f_1, f_2)
  $T := \{s \mid f_2 \in label(s)\}$;
  for all $s \in T$ **do** $label(s) := label(s) \cup \{\mathbf{E}[f_1\ \mathbf{U}\ f_2]\}$;
  while $T \neq \varnothing$ **do**
    choose $s \in T$;
    $T := T \setminus \{s\}$;
    for all t **such that** $R(t, s)$ **do**
      if $\mathbf{E}[f_1\ \mathbf{U}\ f_2] \notin label(t)$ **and** $f_1 \in label(t)$ **then**
        $label(t) := label(t) \cup \{\mathbf{E}[f_1\ \mathbf{U}\ f_2]\}$;
        $T := T \cup \{t\}$;
      end if;
    end for all;
  end while
end procedure

Рис.: Процедура разметки состояний для формулы $\mathbf{E}[f_1\ \mathbf{U}\ f_2]$

Табличный алгоритм model checking для CTL

Случай $g = \mathbf{EG}\, f_1$ чуть более изощрен. Чтобы исследовать его, необходимо разбить граф на нетривиальные сильно связные компоненты.

Сильно связной компонентой (SCC) C называется наибольший подграф, любая вершина которого достижима из всякой другой вершины этого подграфа по ориентированному пути, целиком содержащемуся в C.

Компонента C считается нетривиальной в том и только том случае, когда она содержит более одной вершины или в точности одну вершину с циклом-петлей, проходящим через нее.

Табличный алгоритм model checking для CTL

Пусть модель M' получена из M за счет удаления из S всех тех состояний, в которых не выполняется f_1, и соответствующего сужения R и L.

Таким образом, мы получаем модель $M' = (S', R', L')$, в которой $S' = \{s \in S \mid M, s \models f_1\}$, $R' = R|_{S' \times S'}$ и $L' = L|_{S'}$.

Нужно иметь в виду, что R' после сужения не обязательно будет тотальным отношением. Состояния, из которых не исходит ни одного перехода, могут быть удалены, но, вообще говоря, это не сказывается на корректности алгоритма.

Сам алгоритм обусловлен следующим утверждением.

Табличный алгоритм model checking для CTL

Лемма 1. $M, s \models \mathbf{EG}\, f_1$ тогда и только тогда, когда соблюдены следующие два условия:

- состояние s содержится в множестве S';
- в M' есть путь, ведущий из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').

Табличный алгоритм model checking для CTL

Лемма 1. $M, s \models \mathbf{EG}\, f_1$ тогда и только тогда, когда соблюдены следующие два условия:

- состояние s содержится в множестве S';
- в M' есть путь, ведущий из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').

Доказательство. Допустим, что $M, s \models \mathbf{EG}\, f_1$. Ясно, что $s \in S'$. Рассмотрим такой бесконечный путь π, исходящий из s, что f_1 выполняется в каждом состоянии на протяжении π.

Табличный алгоритм model checking для CTL

Лемма 1. $M, s \models \mathbf{EG}\, f_1$ тогда и только тогда, когда соблюдены следующие два условия:

- состояние s содержится в множестве S';
- в M' есть путь, ведущий из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').

Доказательство. Допустим, что $M, s \models \mathbf{EG}\, f_1$. Ясно, что $s \in S'$. Рассмотрим такой бесконечный путь π, исходящий из s, что f_1 выполняется в каждом состоянии на протяжении π. Так как модель M конечна, путь π всегда можно представить в виде $\pi = \pi_0\pi_1$, где π_0 — конечный начальный отрезок, а π_1 — бесконечный суффикс π, обладающий тем свойством, что каждое состояние в π_1 встречается в нем бесконечно часто. Ясно, что π_0 содержится в S'.

Табличный алгоритм model checking для CTL

Лемма 1. $M, s \models \mathbf{EG}\, f_1$ тогда и только тогда, когда соблюдены следующие два условия:

- состояние s содержится в множестве S';
- в M' есть путь, ведущий из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').

Доказательство. Допустим, что $M, s \models \mathbf{EG}\, f_1$. Ясно, что $s \in S'$. Рассмотрим такой бесконечный путь π, исходящий из s, что f_1 выполняется в каждом состоянии на протяжении π. Так как модель M конечна, путь π всегда можно представить в виде $\pi = \pi_0\pi_1$, где π_0 — конечный начальный отрезок, а π_1 — бесконечный суффикс π, обладающий тем свойством, что каждое состояние в π_1 встречается в нем бесконечно часто. Ясно, что π_0 содержится в S'.
Обозначим символом C множество состояний, через которые проходит π_1. Очевидно, что C содержится в S'.

Табличный алгоритм model checking для CTL

Доказательство.

Покажем, что между любой парой состояний из C проходит путь, целиком лежащий в C.

Рассмотрим произвольные состояния s_1 и s_2 из C. Выберем некоторое вхождение s_1 в π_1. Учитывая то, каким образом был выбран путь π_1, мы можем утверждать, что одно из вхождений состояния s_2 располагается на пути π_1 еще дальше.

Отрезок между выбранными вхождениями состояний s_1 и s_2 целиком лежит в C. Этот отрезок представляет собой путь из s_1 в s_2, лежащий в C.

Таким образом, C либо является сильно связной компонентой, либо содержится в одной из таких компонент. В любом случае оба заявленных условия леммы выполняются.

Табличный алгоритм model checking для CTL

Лемма 1. $M, s \models \mathbf{EG}\, f_1$ тогда и только тогда, когда соблюдены следующие два условия:

- состояние s содержится в множестве S';
- в M' есть путь, ведущий из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').

Табличный алгоритм model checking для CTL

Лемма 1. $M, s \models \mathbf{EG}\, f_1$ тогда и только тогда, когда соблюдены следующие два условия:

- состояние s содержится в множестве S';
- в M' есть путь, ведущий из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').

Доказательство.

А теперь допустим, что условия леммы соблюдены.

Рассмотрим путь π_0 из s в t. Обозначим π_1 конечный путь длины ≥ 1, исходящий из t и вновь возвращающийся в t. Такой путь π_1 наверняка существует, т.к. состояние t лежит в нетривиальной сильно связной компоненте.

Все состояния в бесконечном пути $\pi = \pi_0 \pi_1^\omega$ удовлетворяют f_1. Поскольку π — один из допустимых путей, исходящих из s в модели M, мы убеждаемся в том, что $M, s \models \mathbf{EG}\, f_1$. $\square$

Табличный алгоритм model checking для CTL

Алгоритм для случая $g = \mathbf{EG}\, f_1$ вытекает из леммы.

Табличный алгоритм model checking для CTL

Алгоритм для случая $g = \mathbf{EG}\, f_1$ вытекает из леммы.

Вначале строим сокращенную модель Крипке $M' = (S', R', L')$, как это было описано выше.

Табличный алгоритм model checking для CTL

Алгоритм для случая $g = \mathbf{EG}\, f_1$ вытекает из леммы.

Вначале строим сокращенную модель Крипке $M' = (S', R', L')$, как это было описано выше.

Далее разбиваем граф (S', R') на сильно связные компоненты, применяя один из эффективных алгоритмов решения этой задачи (например, алгоритм Тарьяна). Этот алгоритм имеет сложность $O(|S'| + |R'|)$.

Табличный алгоритм model checking для CTL

Алгоритм для случая $g = \mathbf{EG}\, f_1$ вытекает из леммы.

Вначале строим сокращенную модель Крипке $M' = (S', R', L')$, как это было описано выше.

Далее разбиваем граф (S', R') на сильно связные компоненты, применяя один из эффективных алгоритмов решения этой задачи (например, алгоритм Тарьяна). Этот алгоритм имеет сложность $O(|S'| + |R'|)$.

Затем выделяем те состояния, которые принадлежат сильно связным компонентам. Потом возвращаемся назад, используя обращение R', и выделяем, таким образом, все те состояния, которые лежат на путях, проходящих по состояниям, помеченным f_1.

Табличный алгоритм model checking для CTL

Все вычисление можно выполнить за время $O(|S| + |R|)$.

Процедура *CheckEG*, добавляющая $\mathbf{EG}\, f_1$ к множеству $label(s)$ для каждого s, в котором выполняется $\mathbf{EG}\, f_1$, при условии, что формула f_1 уже была корректно обработана.

Табличный алгоритм model checking для CTL

procedure CheckEG(f_1)
  $S' := \{s \mid f_1 \in label(s)\}$;
  $SCC := \{C \mid C$ — нетривиальная SCC в $S'\}$;
  $T := \bigcup_{C \in SCC} \{s \mid s \in C\}$;
  for all $s \in T$ **do** $label(s) := label(s) \cup \{\mathbf{EG}\, f_1\}$;
  while $T \neq \varnothing$ **do**
    choose $s \in T$;
    $T := T \setminus \{s\}$;
    for all t **such that** $t \in S'$ **and** $R(t, s)$ **do**
      if $\mathbf{EG}\, f_1 \notin label(t)$ **then**
        $label(t) := label(t) \cup \{\mathbf{EG}\, f_1\}$;
        $T := T \cup \{t\}$;
      end if;
    end for all;
  end while
end procedure

Рис.: Процедура разметки состояний для формулы $\mathbf{EG}\, f_1$

Табличный алгоритм model checking для CTL

Чтобы проверить CTL-формулу φ, нужно применять алгоритм разметки состояний ко всем ее подформулам, начиная с самых коротких и наиболее глубоко вложенных подформул, и продвигаться наружу, пока не будет охвачена формула φ.

Следуя такой схеме вычисления, можно быть уверенным в том, что как только мы принимаемся за некоторую подформулу формулы φ, все ее подформулы уже были обработаны.

Коль скоро каждый проход требует времени $O(|S| + |R|)$, а формула φ может содержать не более $|\varphi|$ различных подформул, вся работа алгоритма проводится за время $O(|\varphi|(|S| + |R|))$.

Табличный алгоритм model checking для CTL

Теорема. Существует алгоритм проверки выполнимости произвольной CTL-формулы φ в состоянии s на модели $M = (S, R, L)$ за время $O(|\varphi|(|S| + |R|))$.

Табличный алгоритм model checking для CTL

Поясним работу алгоритма верификации моделей для CTL на примере микроволновой печи.

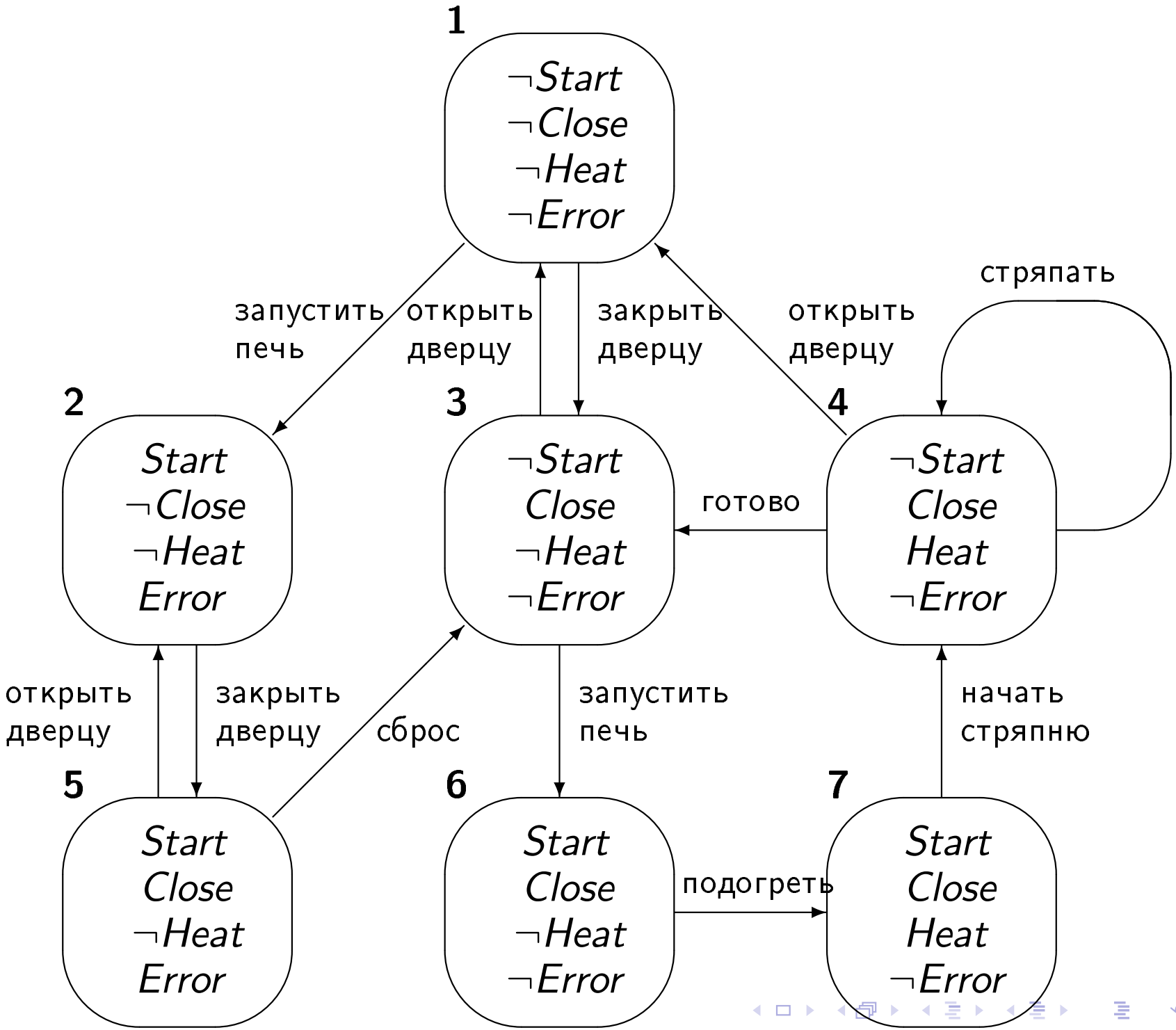
Каждое состояние модели помечено списком, в который включены все те атомарные высказывания, которые истинны в этом состоянии, а также отрицания всех тех атомарных высказываний, которые ложны в этом состоянии. Пометки на дугах обозначают действия, приводящие к переходам.

Проверим на этой модели CTL-формулу

$$\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat}),$$

эквивалентную формуле

$$\varphi = \neg\, \mathbf{E}[\mathit{True}\, \mathbf{U}\, \neg(\neg \mathit{Start} \vee \neg\, \mathbf{EG}\, \neg \mathit{Heat})]$$


1
¬Start
¬Close
¬Heat
¬Error
запустить печь
открыть дверцу
закрыть дверцу
открыть дверцу
стряпать
2
Start
¬Close
¬Heat
Error
3
¬Start
Close
¬Heat
¬Error
готово
4
¬Start
Close
Heat
¬Error
открыть дверцу
закрыть дверцу
сброс
запустить печь
начать стряпню
5
Start
Close
¬Heat
Error
6
Start
Close
¬Heat
¬Error
подогреть
7
Start
Close
Heat
¬Error

Табличный алгоритм model checking для CTL

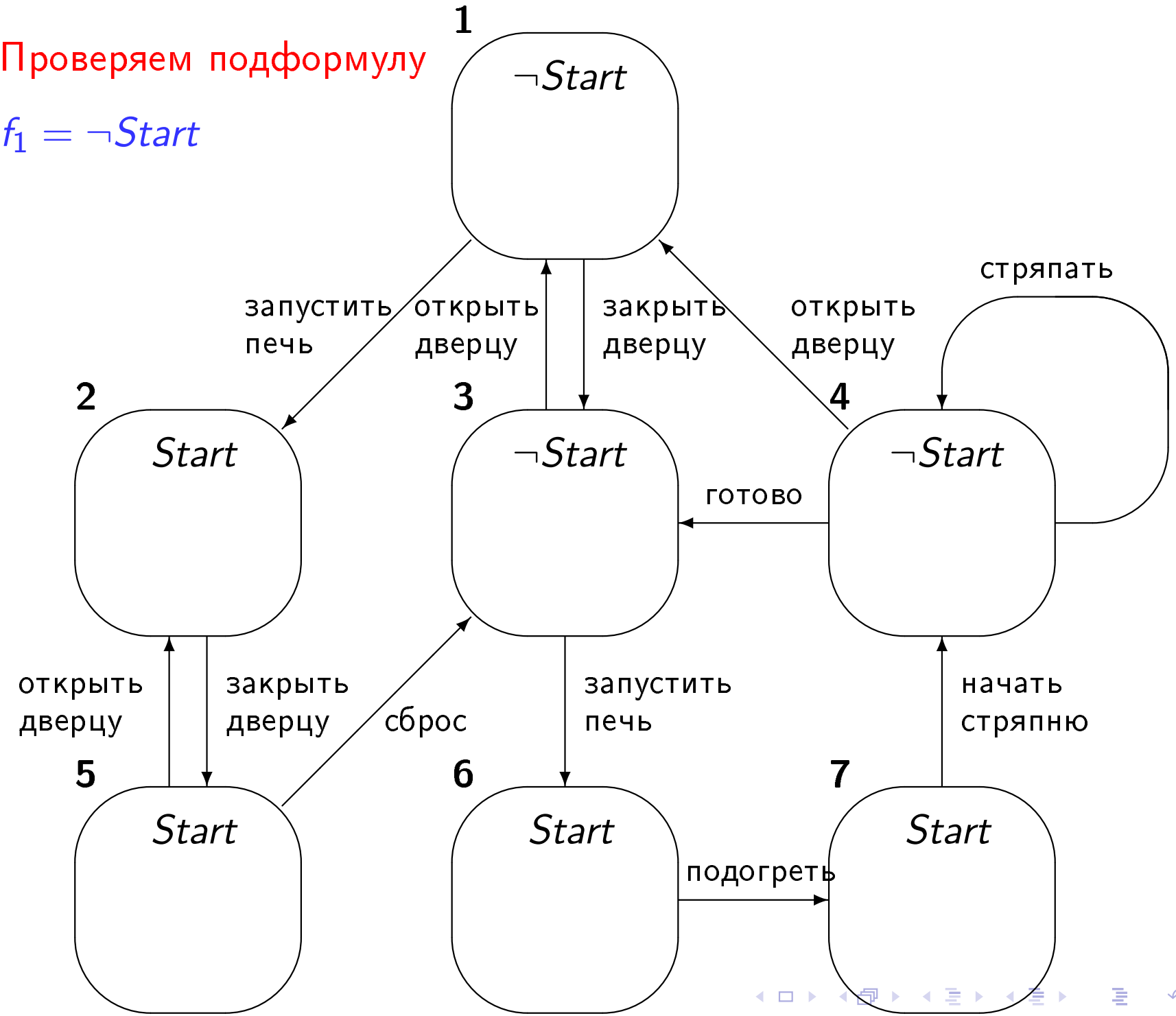
$$\varphi = \neg \mathbf{E}[\mathit{True}\, \mathbf{U}\, \neg(\neg \mathit{Start} \vee \neg\, \mathbf{EG}\, \neg \mathit{Heat})]$$

Выделим подформулы:

- $f_1 = \neg \mathit{Start}$
- $f_2 = \neg \mathit{Heat}$
- $f_3 = \mathbf{EG}\, f_2$
- $f_4 = \neg(f_1 \vee \neg f_2)$
- $f_5 = \mathbf{E}[\mathit{True}\, \mathbf{U}\, f_4]$
- $\varphi = \neg f_5$

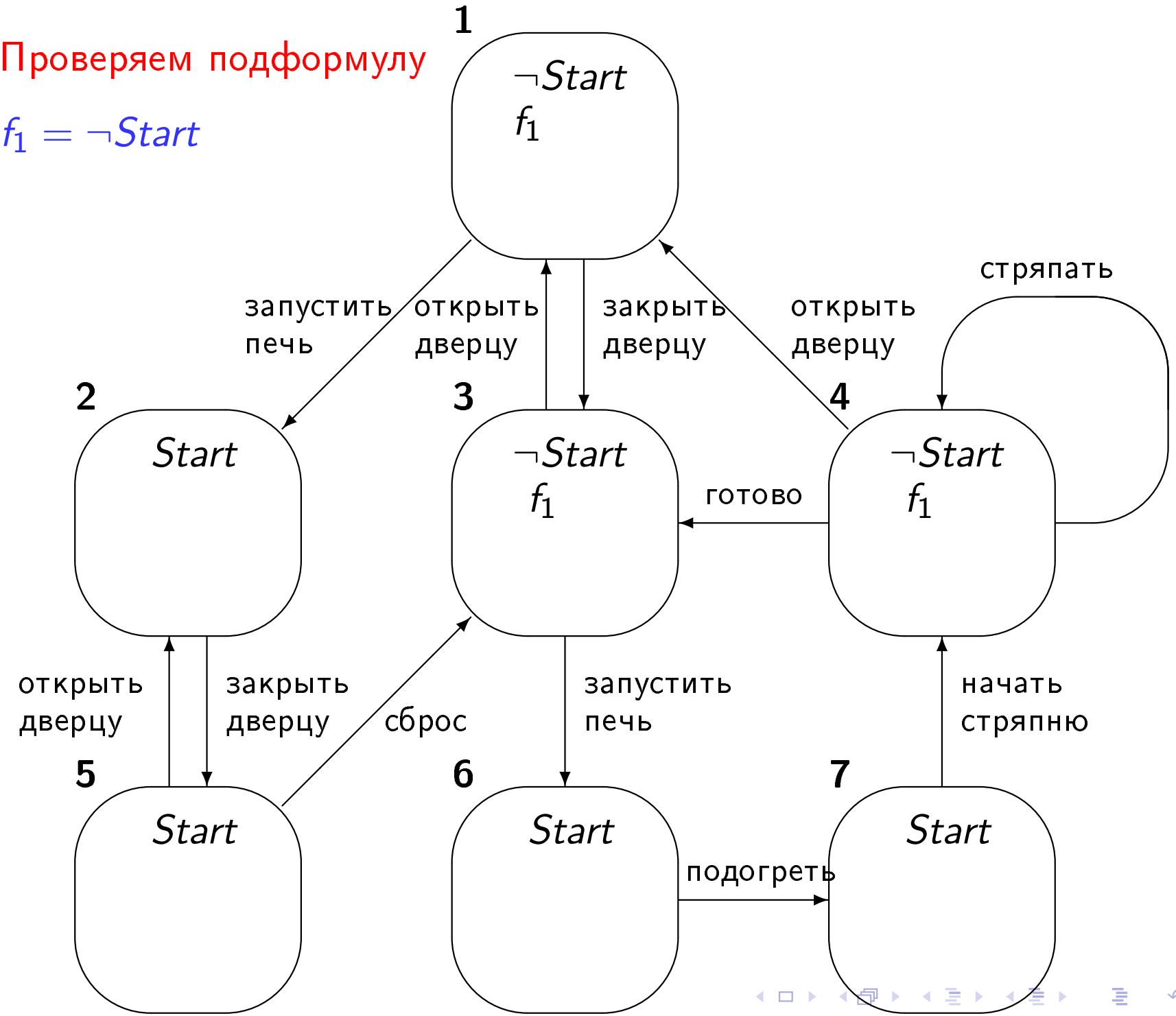
Проверяем подформулу

$f_1 = \neg Start$



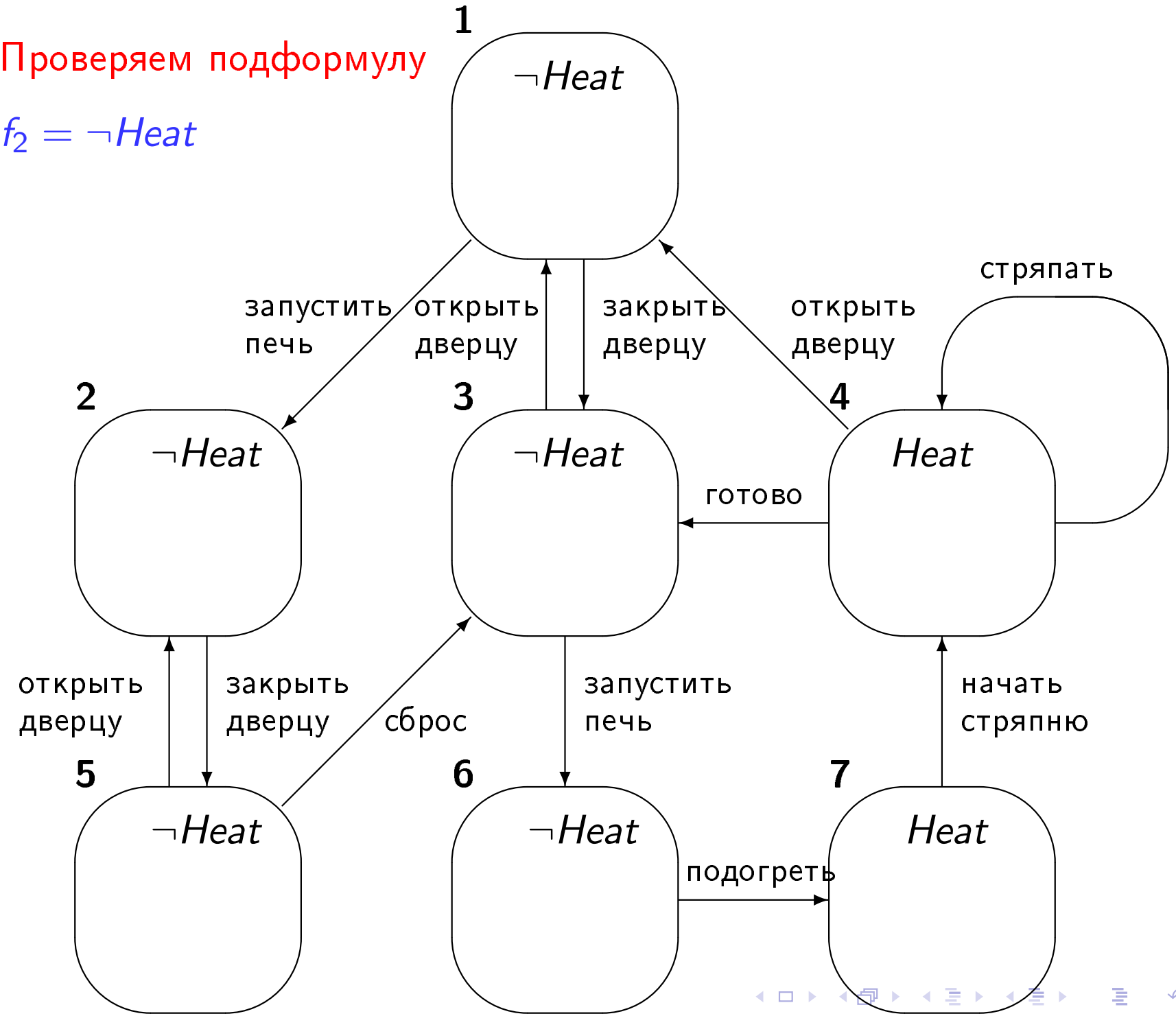
Проверяем подформулу

$f_1 = \neg Start$



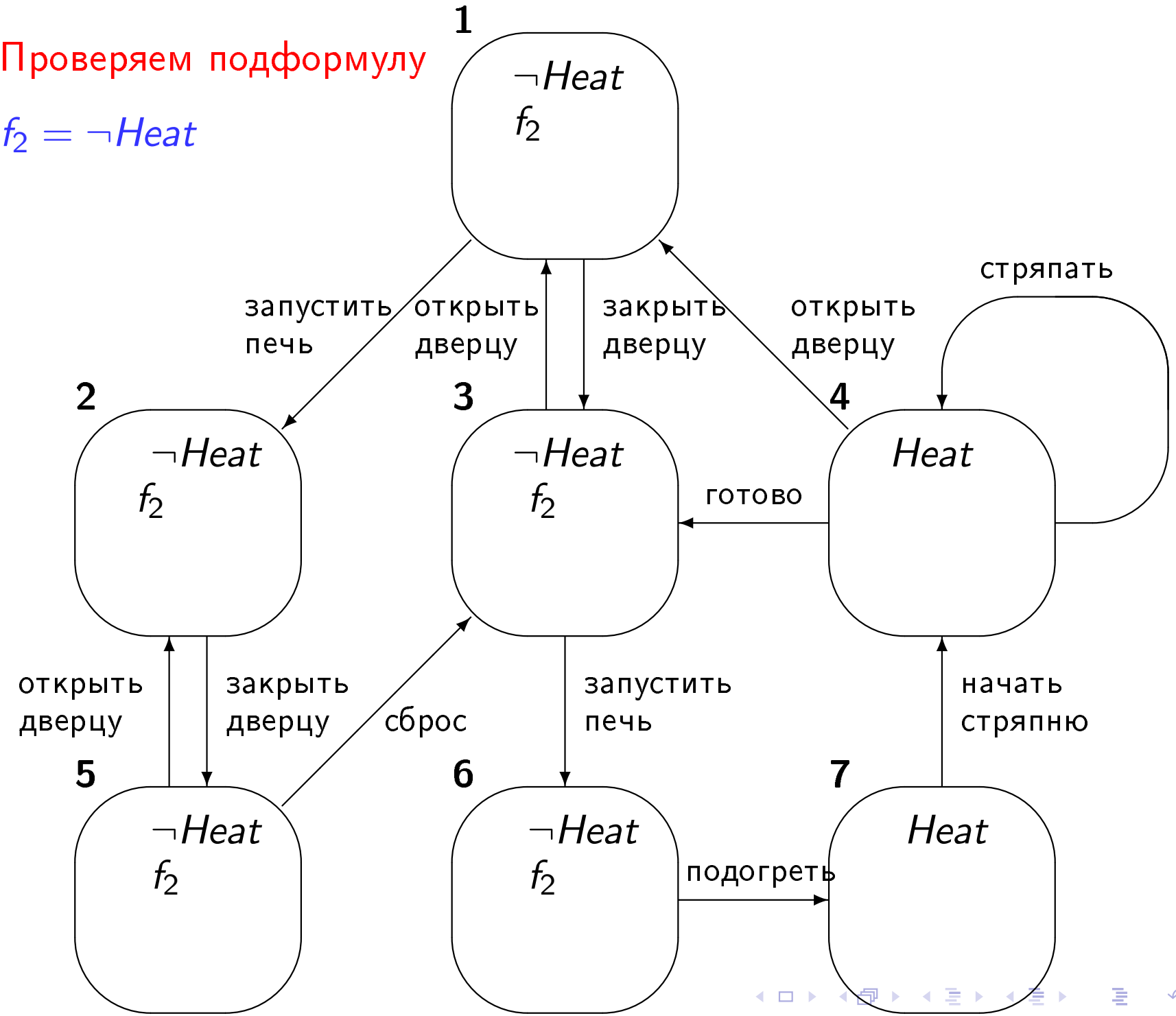
Проверяем подформулу

$f_2 = \neg Heat$



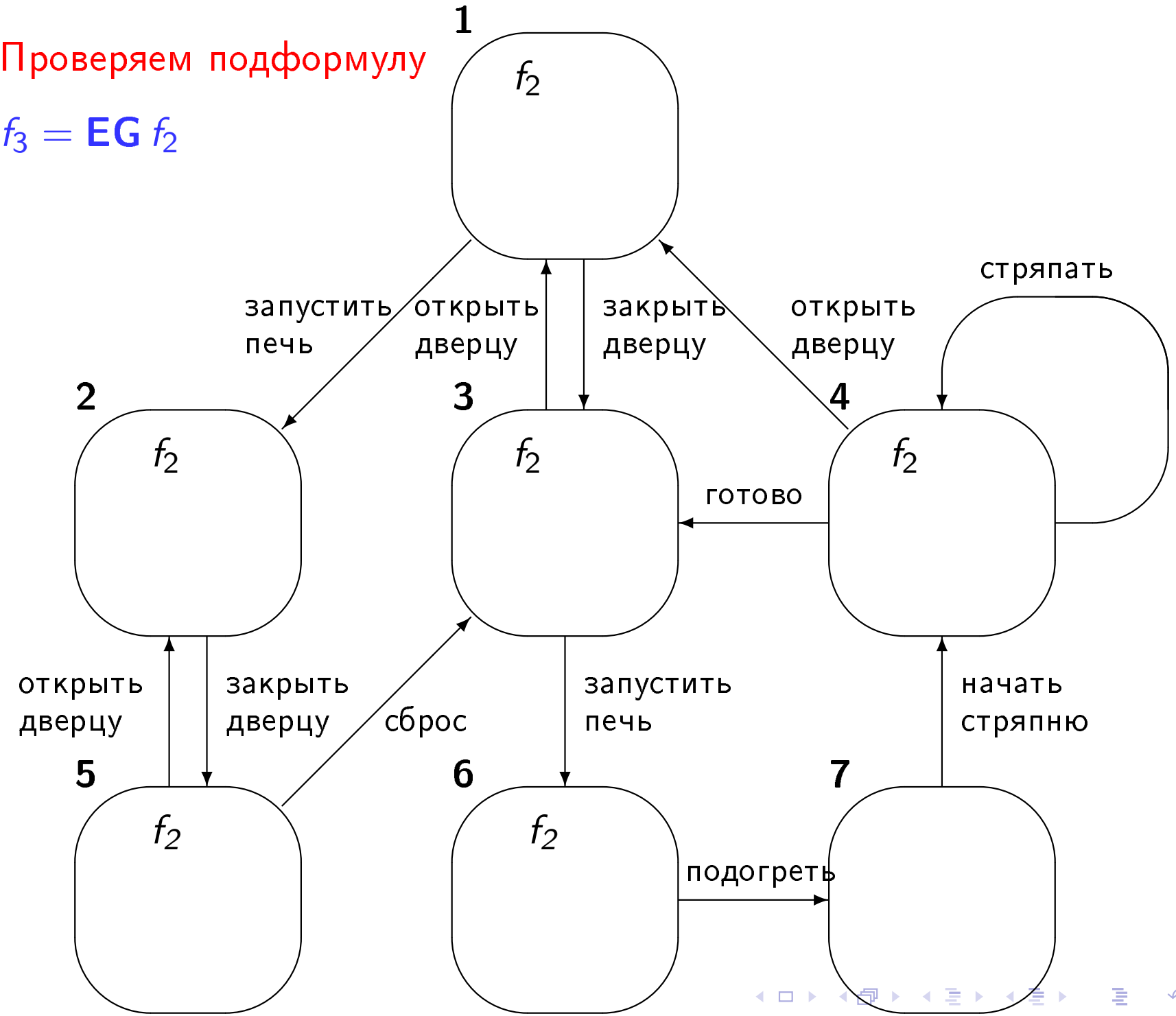
Проверяем подформулу

$f_2 = \neg Heat$



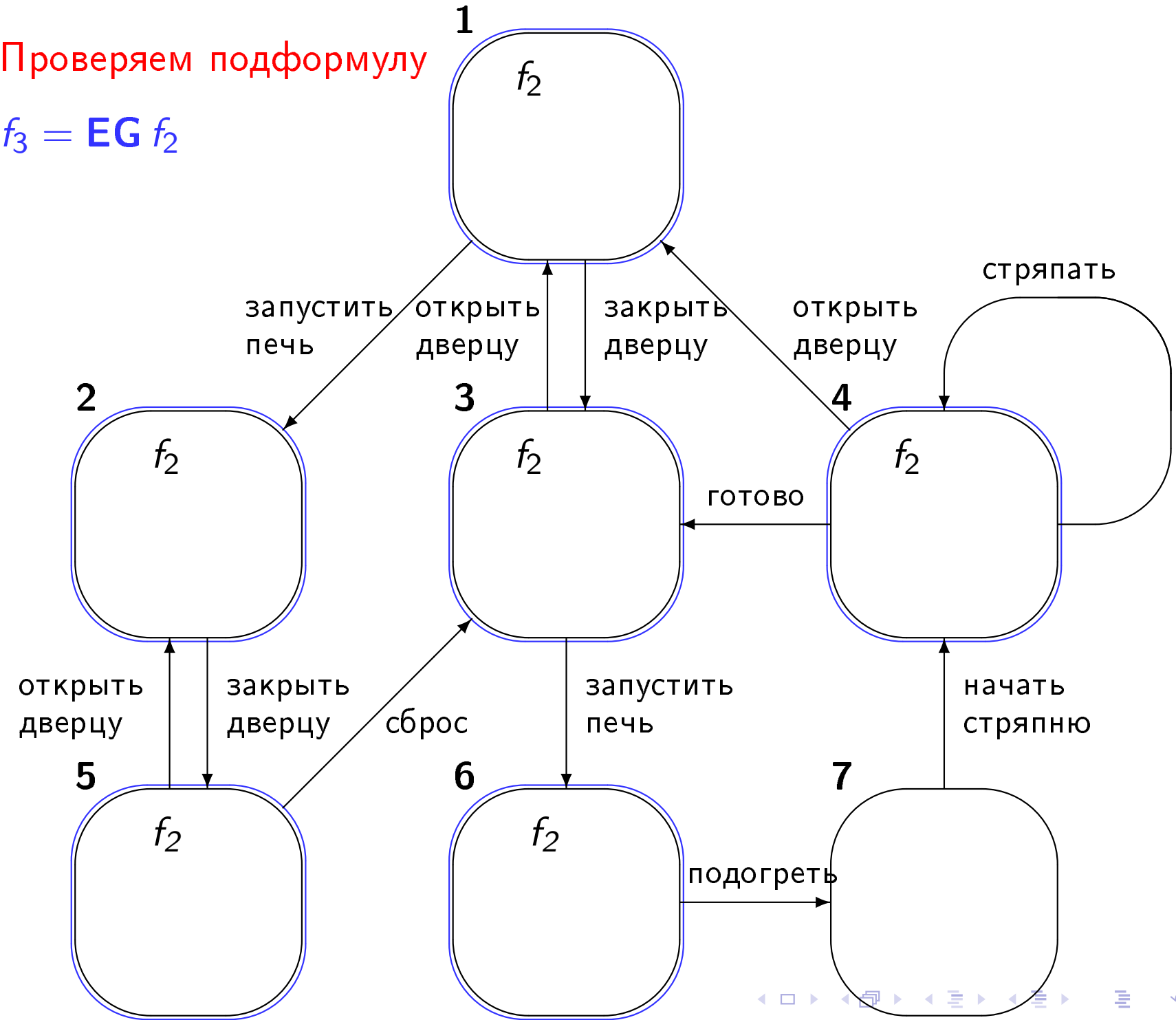
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



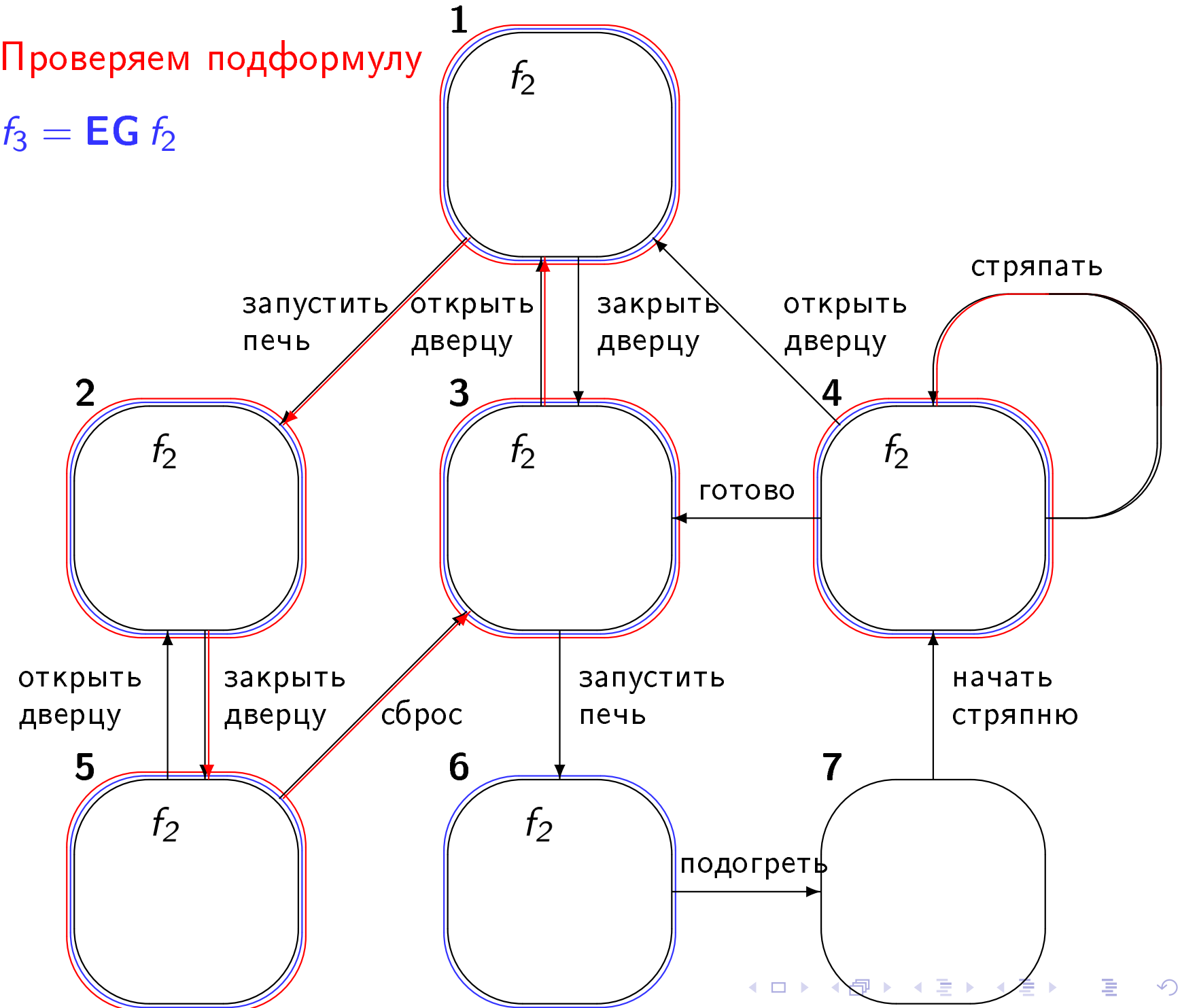
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



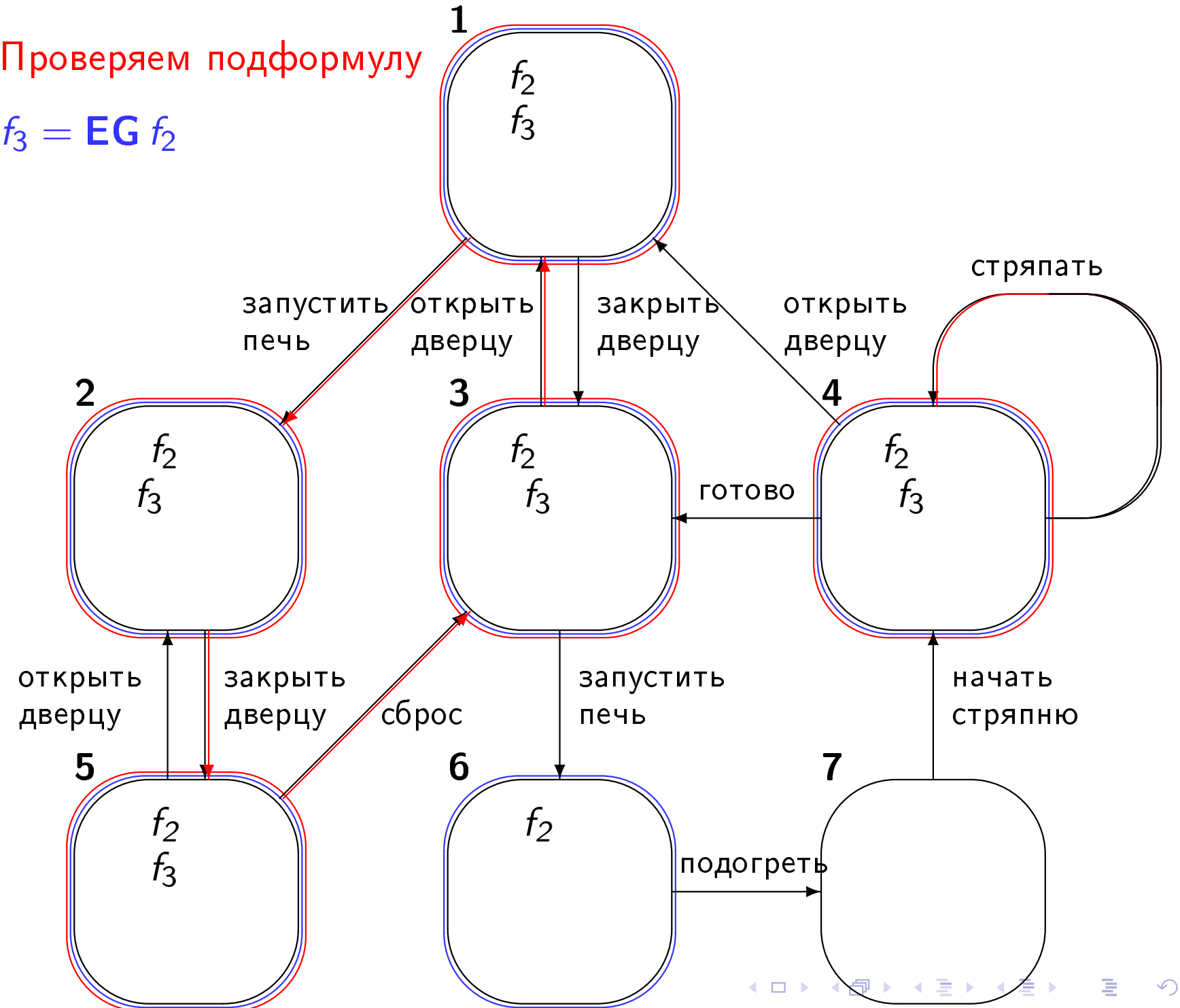
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



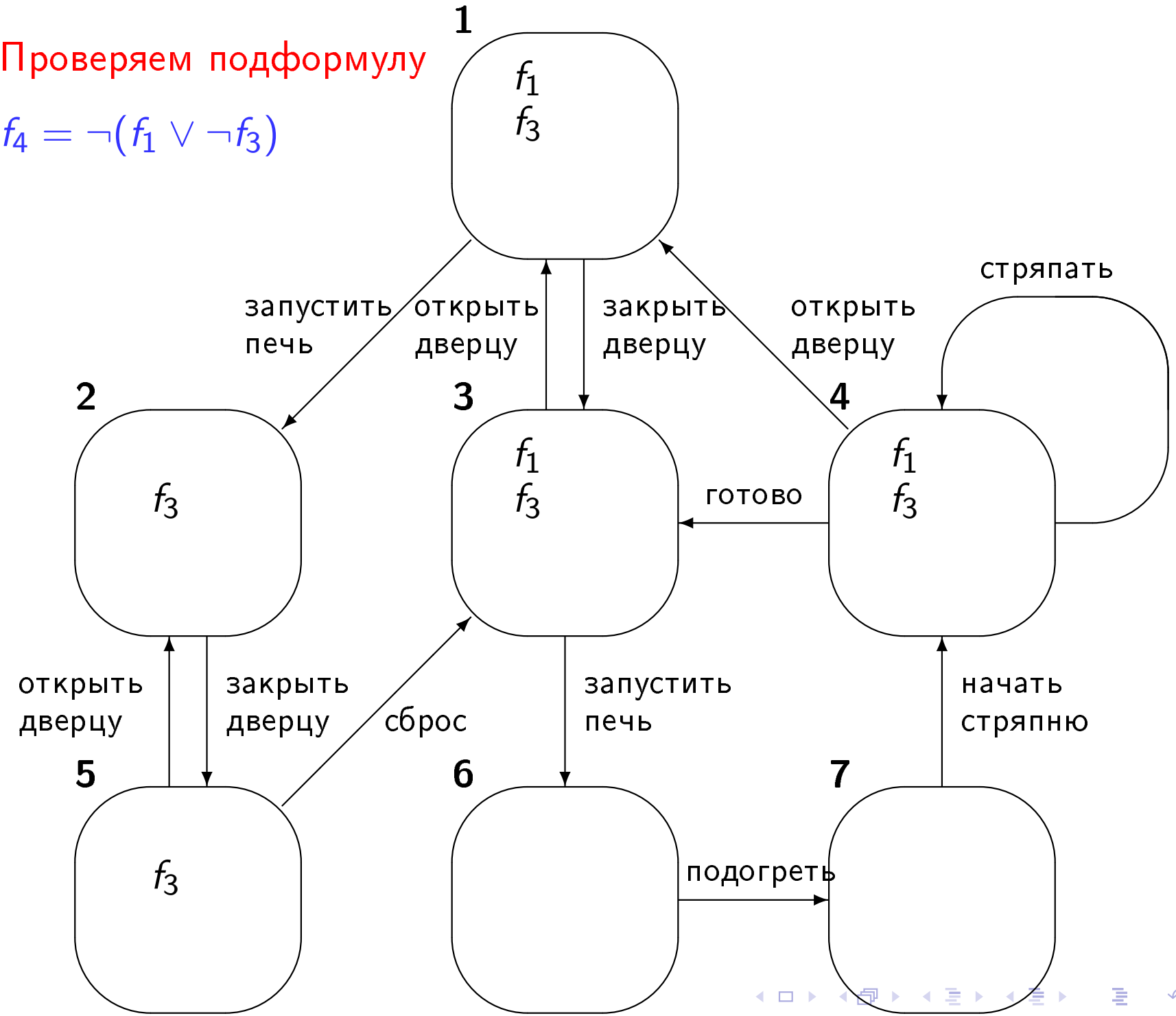
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



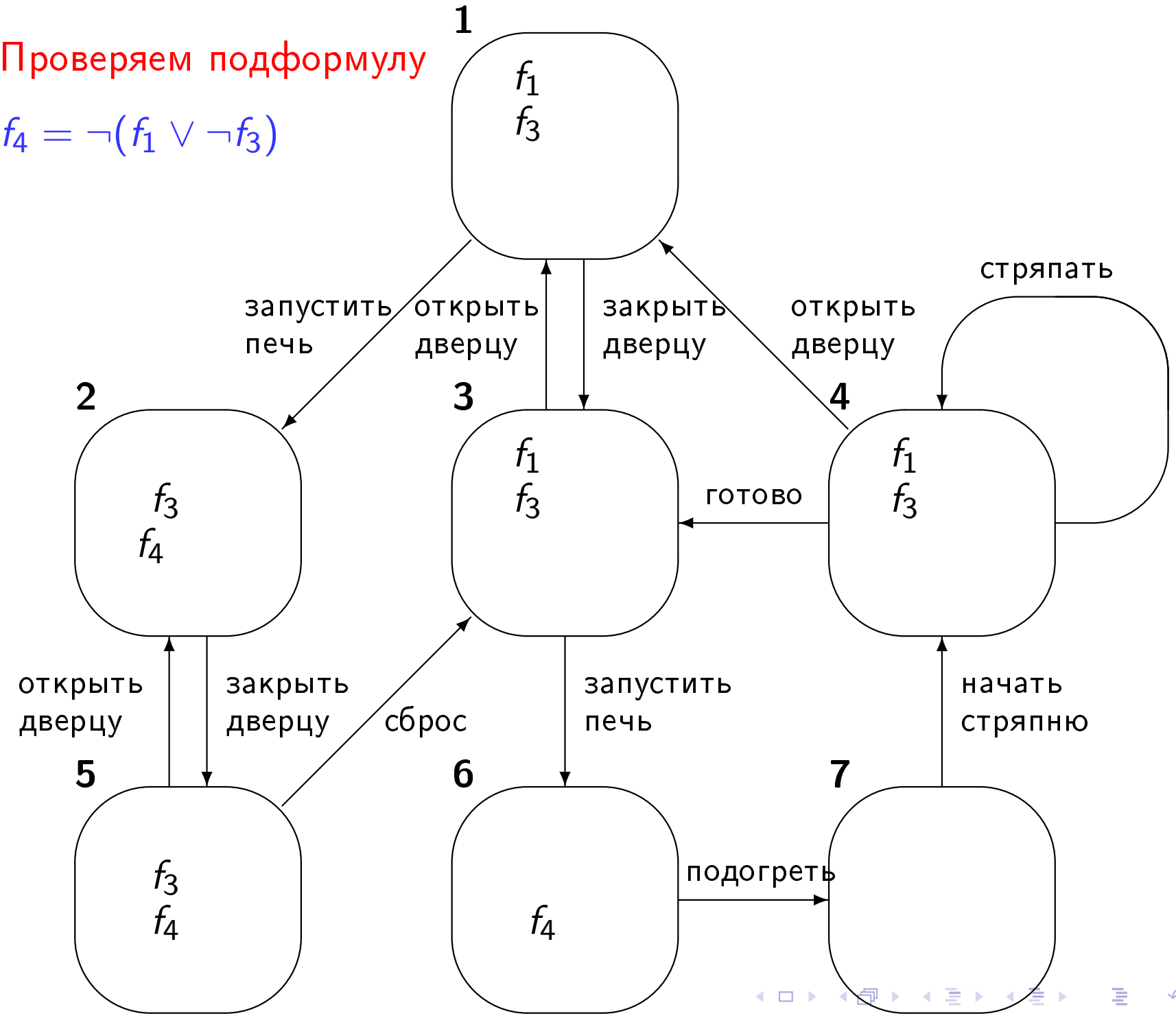
Проверяем подформулу

$f_4 = \neg(f_1 \vee \neg f_3)$



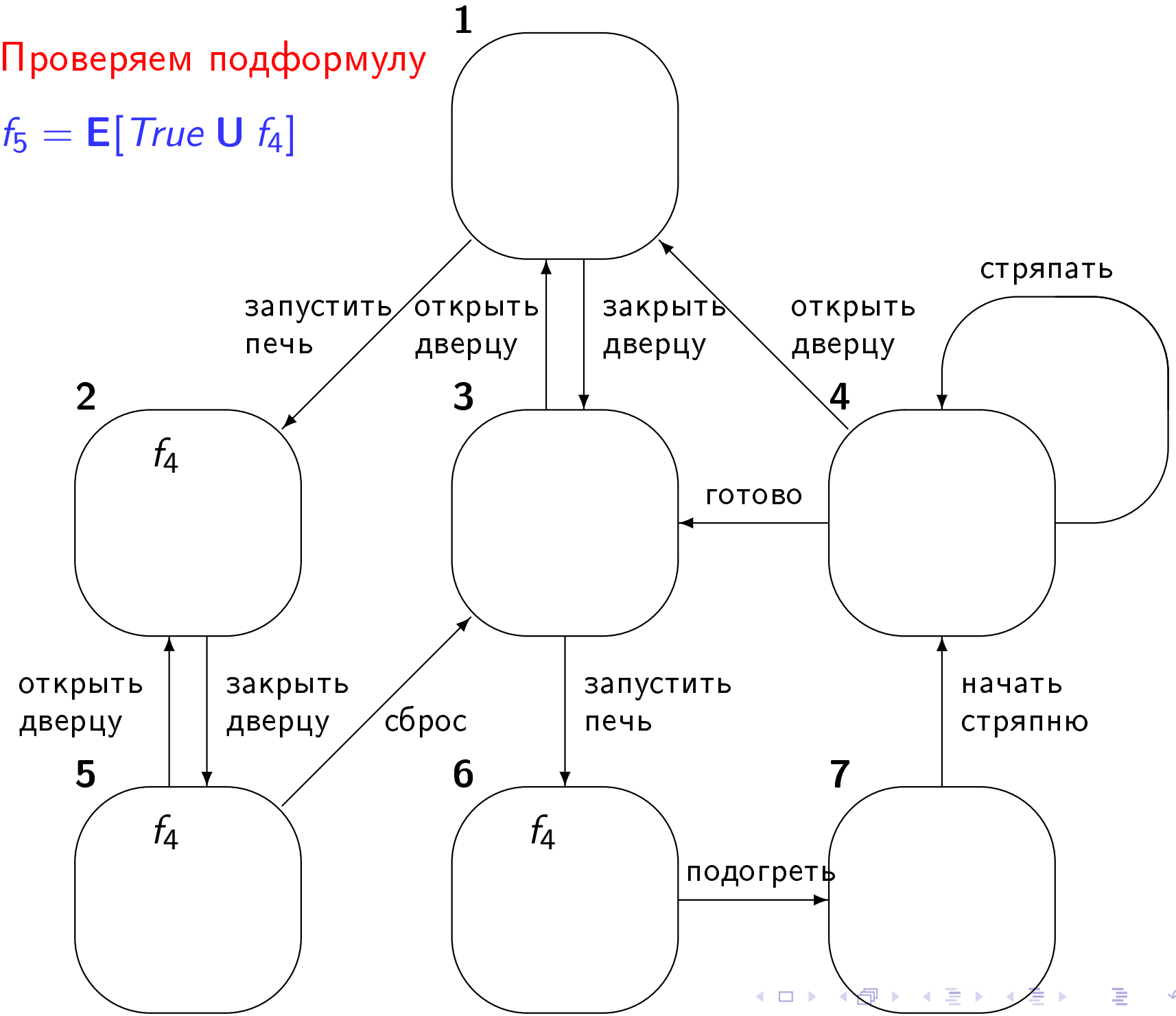
Проверяем подформулу

$f_4 = \neg(f_1 \vee \neg f_3)$



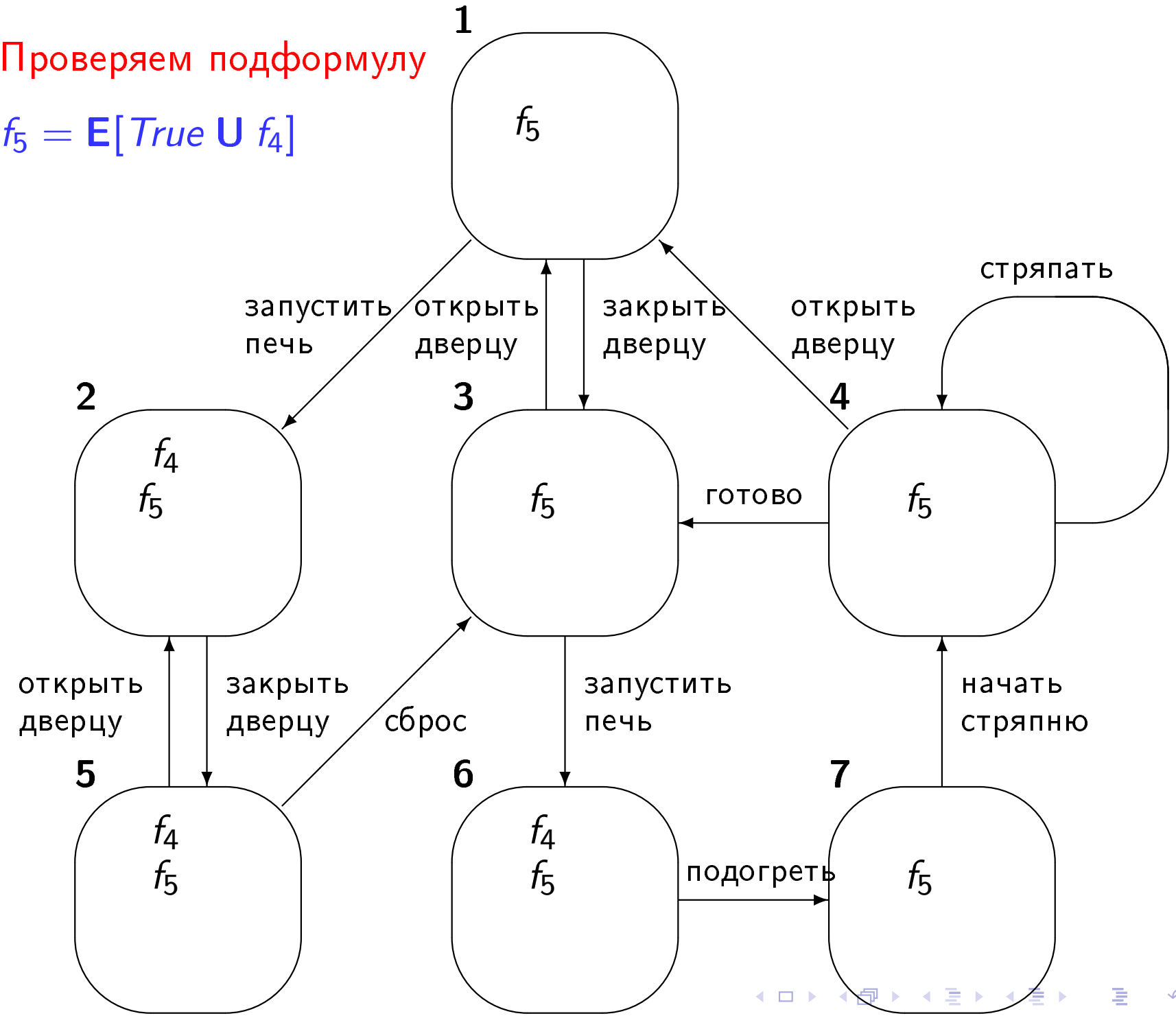
Проверяем подформулу

$f_5 = \mathbf{E}[True\ \mathbf{U}\ f_4]$



Проверяем подформулу

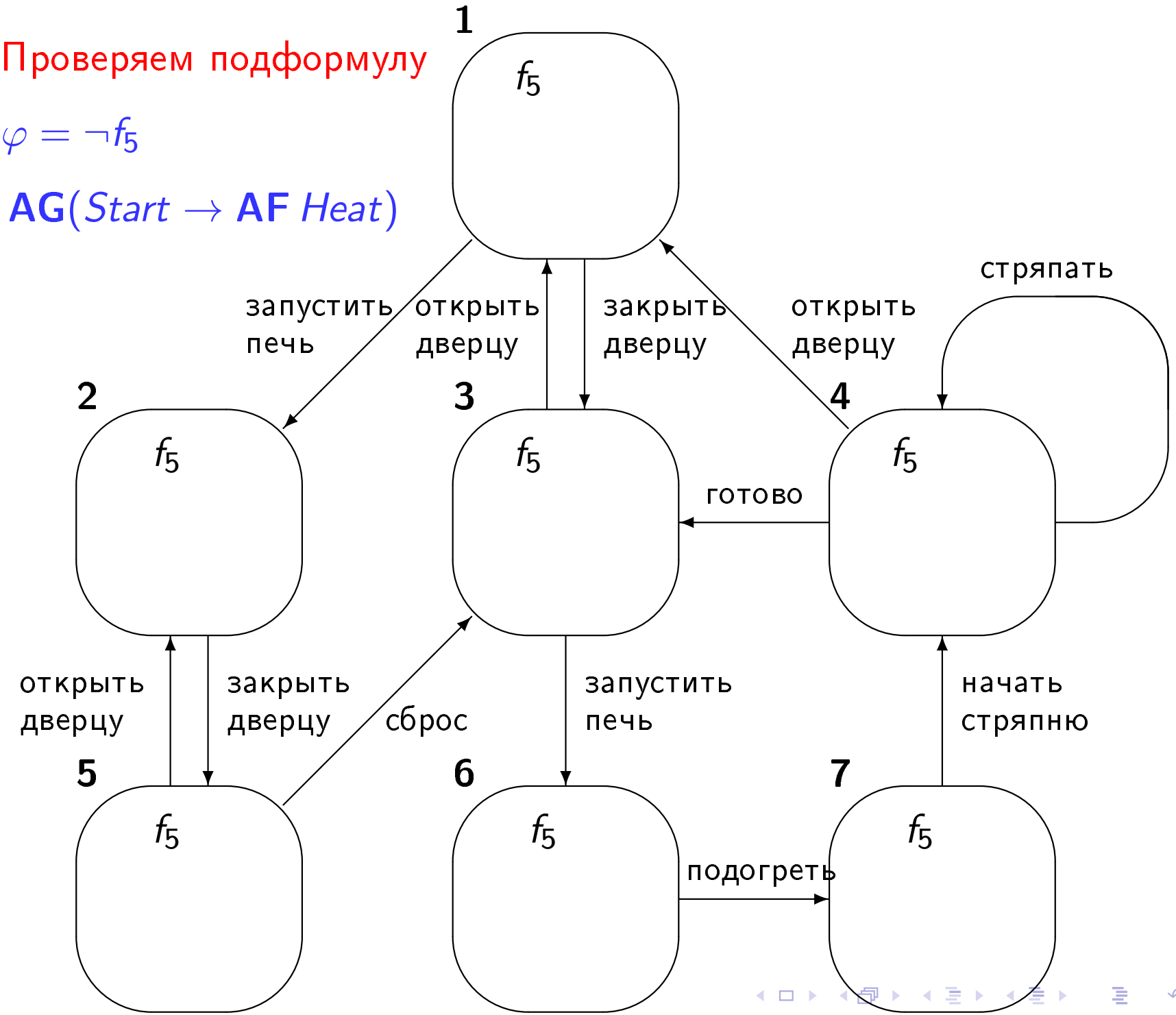
$f_5 = \mathbf{E}[True\ \mathbf{U}\ f_4]$



Проверяем подформулу

$\varphi = \neg f_5$

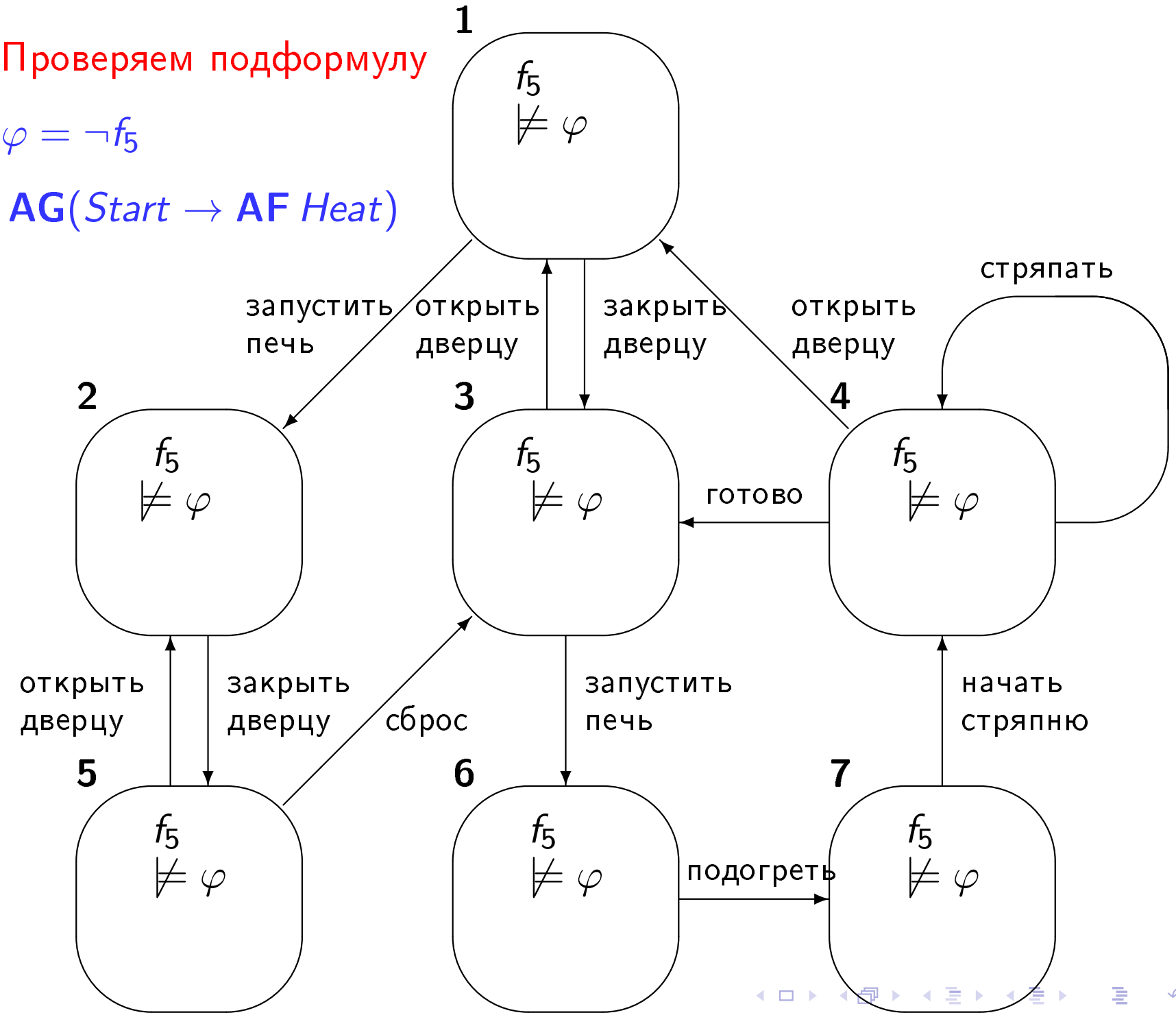
$\varphi = \mathbf{AG}(Start \rightarrow \mathbf{AF}\, Heat)$



Проверяем подформулу

$\varphi = \neg f_5$

$\varphi = \mathbf{AG}(Start \rightarrow \mathbf{AF}\, Heat)$



Табличный алгоритм model checking для CTL

Таким образом, $M, 1 \not\models \mathbf{AG}(\mathit{Start} \rightarrow \mathbf{AF}\, \mathit{Heat})$.

Табличный алгоритм model checking для CTL

Таким образом, $M, 1 \not\models \mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat})$.

Задача. Проверьте выполнимость в модели микроволновой печи выполнимость других спецификаций:

- $\mathbf{AG}(\mathit{Error} \to \mathbf{A}[\neg \mathit{Start}\, \mathbf{R}\, \mathit{Error}])$,
- $\mathbf{AG}\, \mathbf{EX}\, \mathbf{EX}\, \mathbf{EX}\, \mathit{Heat}$,
- $\neg\, \mathbf{EG}(\mathit{Error} \to \mathbf{AX}\, \mathit{Error})$,
- $\mathbf{AG}(\mathbf{A}[\neg \mathit{Start}\, \mathbf{U}\, \mathit{Close}])$.

Model checking для CTL в ограничениях справедливости

Теперь мы покажем, как обобщить алгоритм верификации моделей для логики CTL, чтобы охватить ограничения справедливости.

Model checking для CTL в ограничениях справедливости

Теперь мы покажем, как обобщить алгоритм верификации моделей для логики CTL, чтобы охватить ограничения справедливости.

Рассмотрим справедливую модель Крипке $M = (S, R, L, F)$, в которой $F = \{P_1, \ldots, P_k\}$ — множество ограничений справедливости , где $P_i \subseteq S$ для всех $i, 1 \leq i \leq k$.

Пусть $\pi = s_0, s_1, \ldots$ — путь в M . Введем следующую характеристику пути:

$$inf(\pi) = \{s \mid s = s_i \text{ для бесконечно многих } i\}.$$

Будем говорить, что π — справедливый путь, если для всякого ограничения справедливости P_i из F выполняется соотношение $inf(\pi) \cap P_i \neq \emptyset$.

Model checking для CTL в ограничениях справедливости

Семантика CTL на справедливых моделях Крипке очень похожа на семантику CTL на обычных моделях Крипке.

Мы будем использовать запись $M, s \models_F \varphi$ для обозначения того, что формула состояния φ истинна в состоянии s на справедливой модели Крипке M .
Изменения вносятся только в некоторые пункты определения отношения выполнимости для CTL.

- $M, s \models_F p \Leftrightarrow$ есть F -справедливый путь, исходящий из s , и при этом $p \in L(s)$;
- $M, s \models_F \mathbf{E}(\varphi) \Leftrightarrow$ в модели M есть такой F -справедливый путь π из состояния s , что $M, \pi \models \varphi$;
- $M, s \models_F \mathbf{A}(\varphi) \Leftrightarrow$ для каждого справедливого пути π в модели M из состояния s верно соотношение $M, \pi \models g_1$.

Model checking для CTL в ограничениях справедливости

Будем говорить, что сильно связная компонента C графа M является справедливой относительно ограничений справедливости F , в том и только том случае, когда для каждого $P_i \in F$ существует состояние $t_i \in C \cap P_i$.

Вначале приведем алгоритм для проверки $\mathbf{EG}\, f_1$ на **справедливой** модели.

Model checking для CTL в ограничениях справедливости

procedure CheckFair EG(f_1)
  $S' := \{s \mid f_1 \in label(s)\}$;
  $FairSCC := \{C \mid C$ — нетривиальная справедливая SCC в $S'\}$
  $T := \bigcup_{C \in FairSCC} \{s \mid s \in C\}$;
  for all $s \in T$ **do** $label(s) := label(s) \cup \{\mathbf{EG}\, f_1\}$;
  while $T \neq \varnothing$ **do**
    choose $s \in T$;
    $T := T \setminus \{s\}$;
    for all t **such that** $t \in S'$ **and** $R(t, s)$ **do**
      if $\mathbf{EG}\, f_1 \notin label(t)$ **then**
        $label(t) := label(t) \cup \{\mathbf{EG}\, f_1\}$;
        $T := T \cup \{t\}$;
      end if;
    end for all;
  end while
end procedure

Model checking для CTL в ограничениях справедливости

Чтобы установить корректность этого алгоритма, потребуется лемма, аналогичная лемме 1.
Как и прежде, будем полагать, что модель M' получена из M за счет удаления из S всех тех состояний, в которых f_1 не выполняется справедливо .
Таким образом, $M' = (S', R', L', F')$, где
$S' = \{s \in S \mid M, s \models_F f_1\}$, $R' = R_{S' \times S'}$, $L' = L_{S'}$ и
$F' = \{P_i \cap S' \mid P_i \in F\}$.

Лемма 2. Соотношение $M, s \models_F \mathbf{EG}\, f_1$ верно тогда и только тогда, когда соблюдены следующие два условия:

- состояние s содержится в множестве S' ,
- в M' имеется путь, ведущий из s в некоторую вершину t , содержащуюся в нетривиальной справедливой сильно связной компоненте C графа (S', R') .

Model checking для CTL в ограничениях справедливости

Для проверки других CTL-формул на справедливых моделях Крипке введем в рассмотрение вспомогательное атомарное высказывание $fair = \mathbf{EG}\, true$. Нетрудно заметить, что для любого состояния s верно

$$M, s \models_F fair \Leftrightarrow \text{из } s \text{ исходит справедливый путь.}$$

Разметить состояния этим новым атомарным высказыванием можно при помощи процедуры $CheckFairEG(true)$.

Model checking для CTL в ограничениях справедливости

Для проверки других CTL-формул на справедливых моделях Крипке введем в рассмотрение вспомогательное атомарное высказывание $fair = \mathbf{EG}\, true$. Нетрудно заметить, что для любого состояния s верно

$$M, s \models_F fair \;\Leftrightarrow\; \text{из}\;\; s \;\; \text{исходит справедливый путь.}$$

Разметить состояния этим новым атомарным высказыванием можно при помощи процедуры $CheckFairEG(true)$.

Чтобы проверить соотношение $M, s \models_F p$ для некоторого $p \in AP$, проверяем соотношение $M, s \models p \wedge fair$ при помощи обычной процедуры верификации моделей.

Для проверки соотношения $M, s \models_F \mathbf{EX}\, f_1$ обращаемся к задаче анализа соотношения $M, s \models \mathbf{EX}(f_1 \wedge fair)$.

И, наконец, чтобы проверить соотношение $M, s \models_F \mathbf{E}[f_1\ \mathbf{U}\ f_2]$ анализируем соотношение $M, s \models \mathbf{E}[f_1\ \mathbf{U}\ (f_2 \wedge fair)]$, обратившись к процедуре $CheckEU(f_1, f_2 \wedge fair)$.

Model checking для CTL в ограничениях справедливости

Теорема. Существует алгоритм проверки выполнимости произвольной CTL-формулы φ в состоянии s на модели $M = (S, R, L, F)$ в рамках справедливой семантики за время $O(|\varphi|(|S| + |R|)|F|)$.

Анализ сложности таков же, как и при отсутствии справедливости.

Время, необходимое на каждом этапе работы алгоритма, составляет $O((|S| + |R|)|F|)$.

Так как число таких этапов не превосходит $|\varphi|$, общая сложность по времени равна $O(|\varphi|(|S| + |R|)|F|)$.

Model checking для CTL в ограничениях справедливости

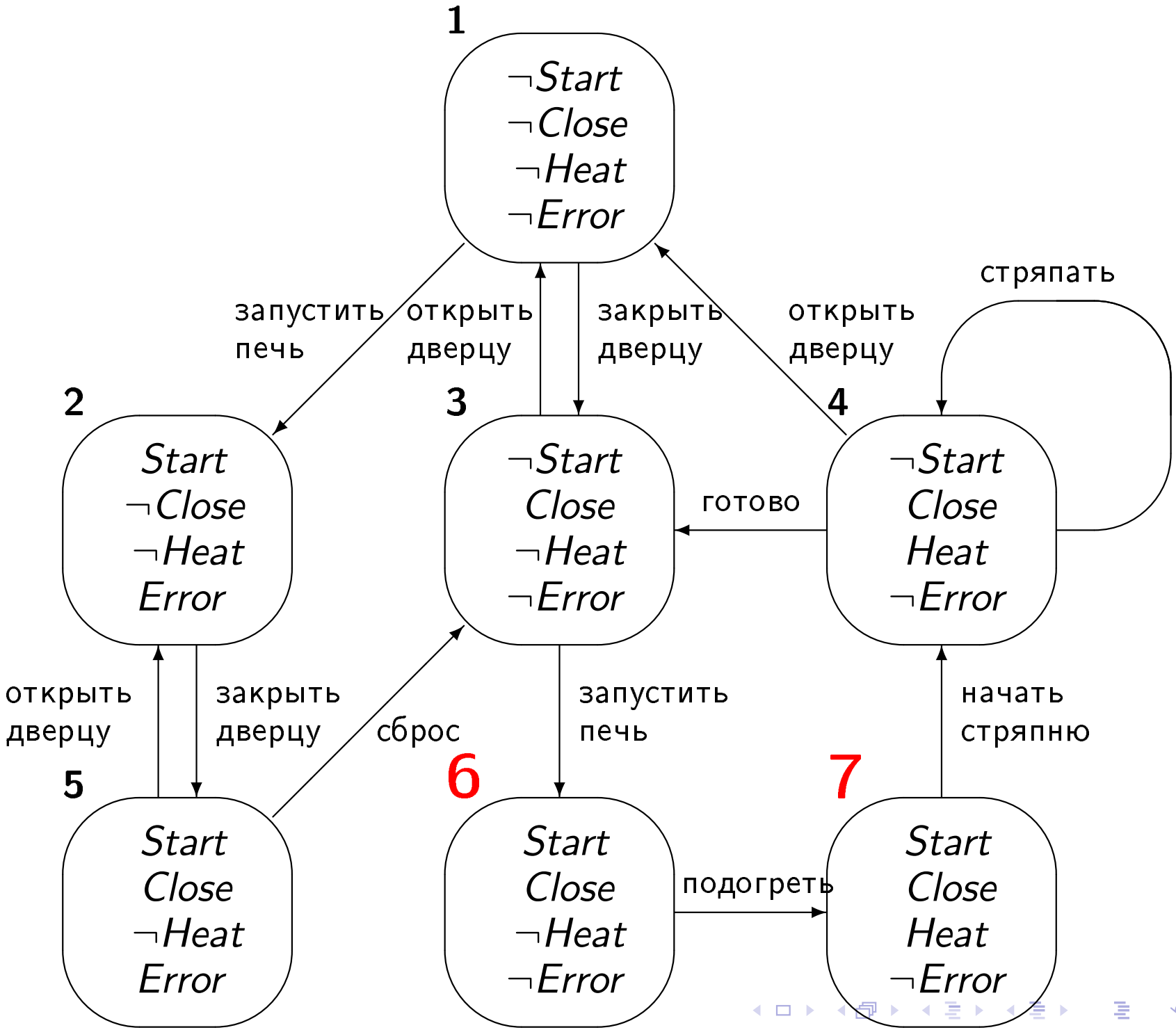
Чтобы посмотреть, какое влияние оказывает справедливость, обратимся вновь к формуле $\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\, \mathit{Heat})$ или, к равносильной ей формуле

$$\varphi = \neg\, \mathbf{E}[\mathit{True}\, \mathbf{U}\, (\mathit{Start} \wedge \mathbf{EG}\, \neg \mathit{Heat})]$$

на модели микроволновой печи.

Однако на этот раз будем рассматривать только те пути, на протяжении которых пользователь правильно работает с микроволновой печью бесконечно часто. Это означает, что должно бесконечно часто выполняться условие $\mathit{Start} \wedge \mathit{Close} \wedge \neg \mathit{Error}$.

Итак, $F = \{P\}$, где $P = \{s \mid s \models \mathit{Start} \wedge \mathit{Close} \wedge \neg \mathit{Error}\}$.


1
¬Start
¬Close
¬Heat
¬Error
запустить печь
открыть дверцу
закрыть дверцу
открыть дверцу
стряпать
2
Start
¬Close
¬Heat
Error
3
¬Start
Close
¬Heat
¬Error
готово
4
¬Start
Close
Heat
¬Error
открыть дверцу
закрыть дверцу
сброс
запустить печь
начать стряпню
5
Start
Close
¬Heat
Error
6
Start
Close
¬Heat
¬Error
подогреть
7
Start
Close
Heat
¬Error

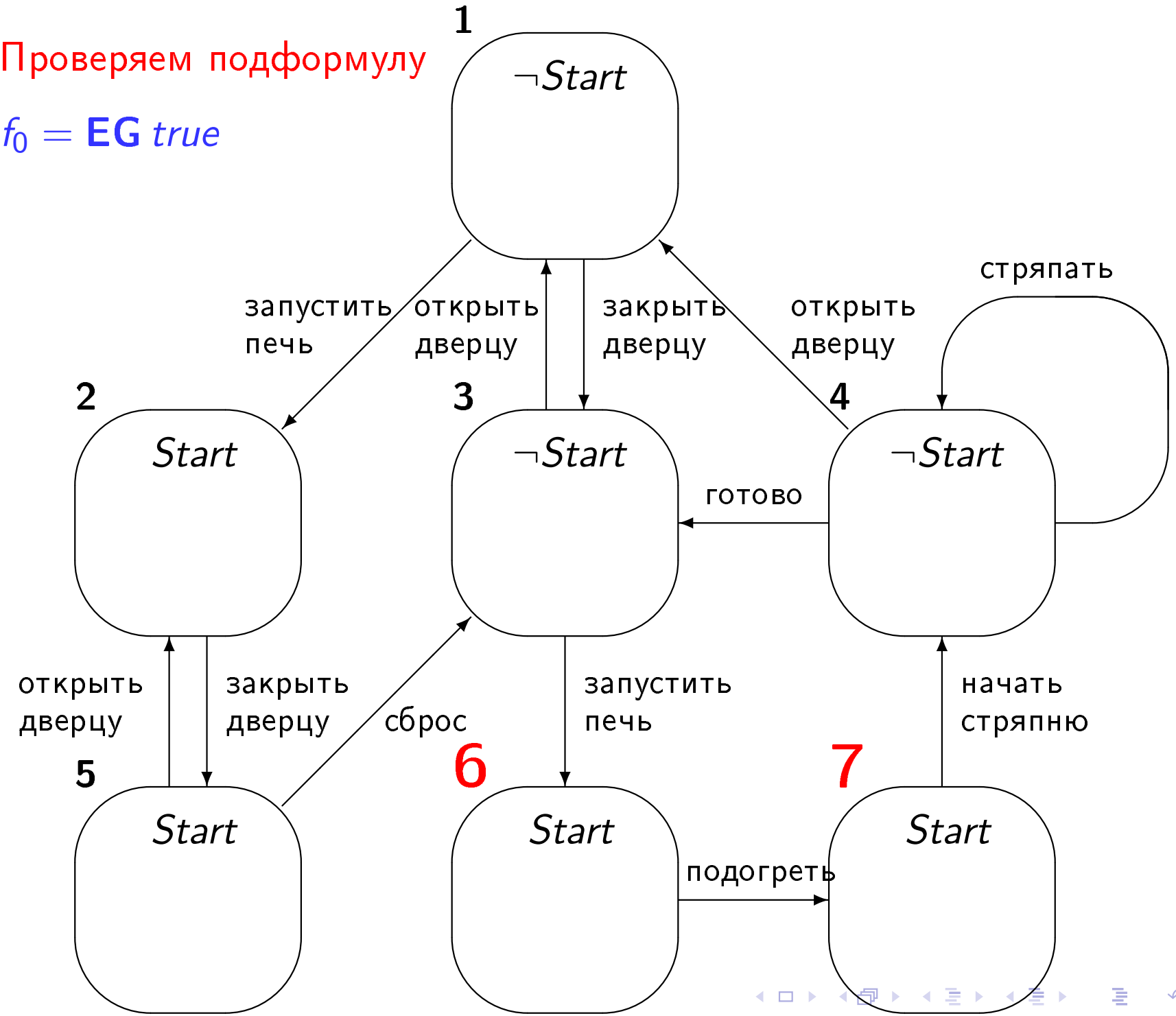
Табличный алгоритм model checking для CTL

Выделим подформулы:

- $f_0 = fair = \mathbf{EG}\, true$
- $f_1 = Start \wedge f_0$
- $f_2 = \neg(Heat \wedge f_0)$
- $f_3 = \mathbf{EG}\, f_2$
- $f_4 = f_1 \wedge f_3$
- $f_5 = \mathbf{E}[True\, \mathbf{U}\, f_4 \wedge f_0]$
- $\varphi = \neg f_5$

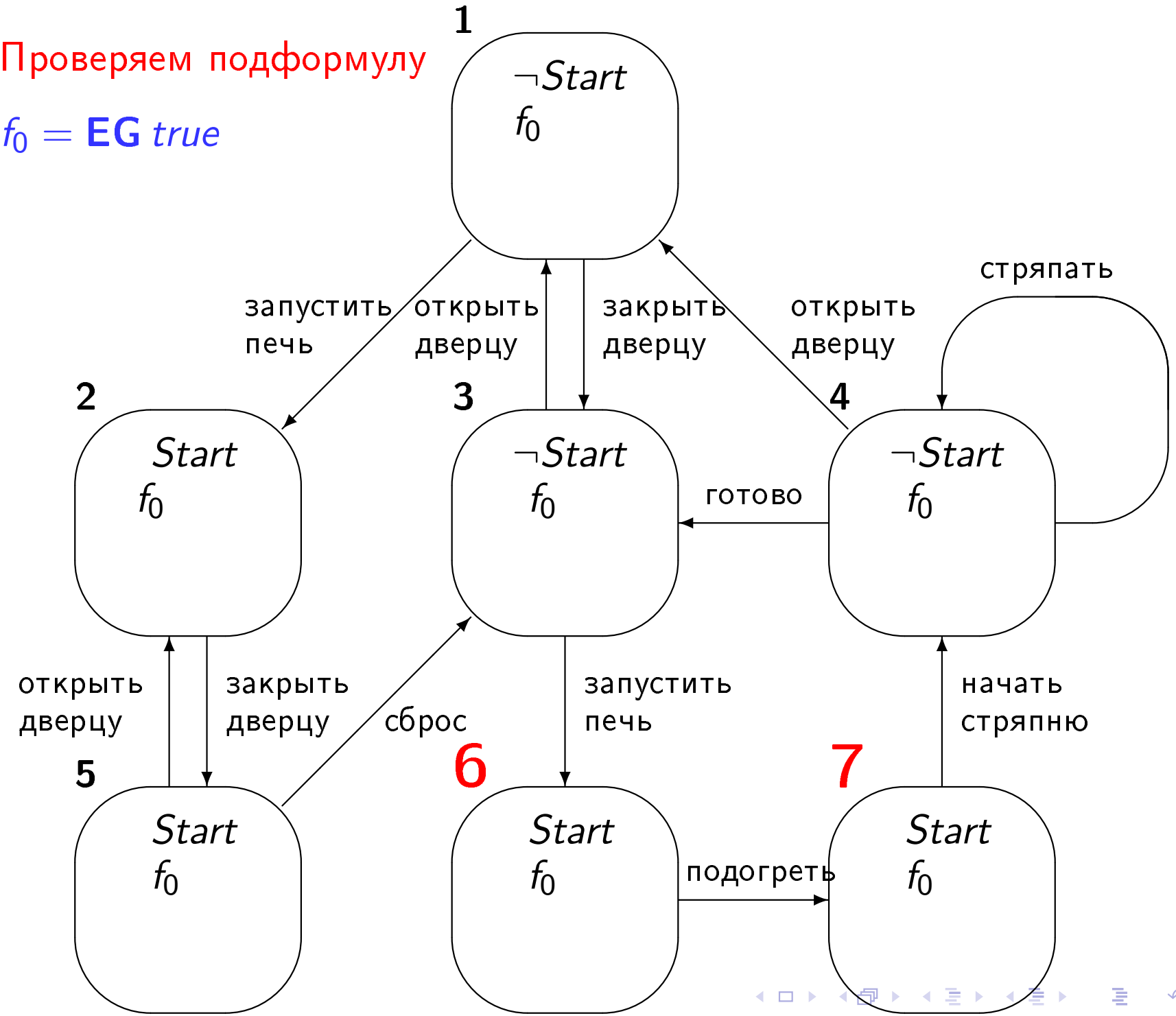
Проверяем подформулу

$f_0 = \mathbf{EG}\ true$



Проверяем подформулу

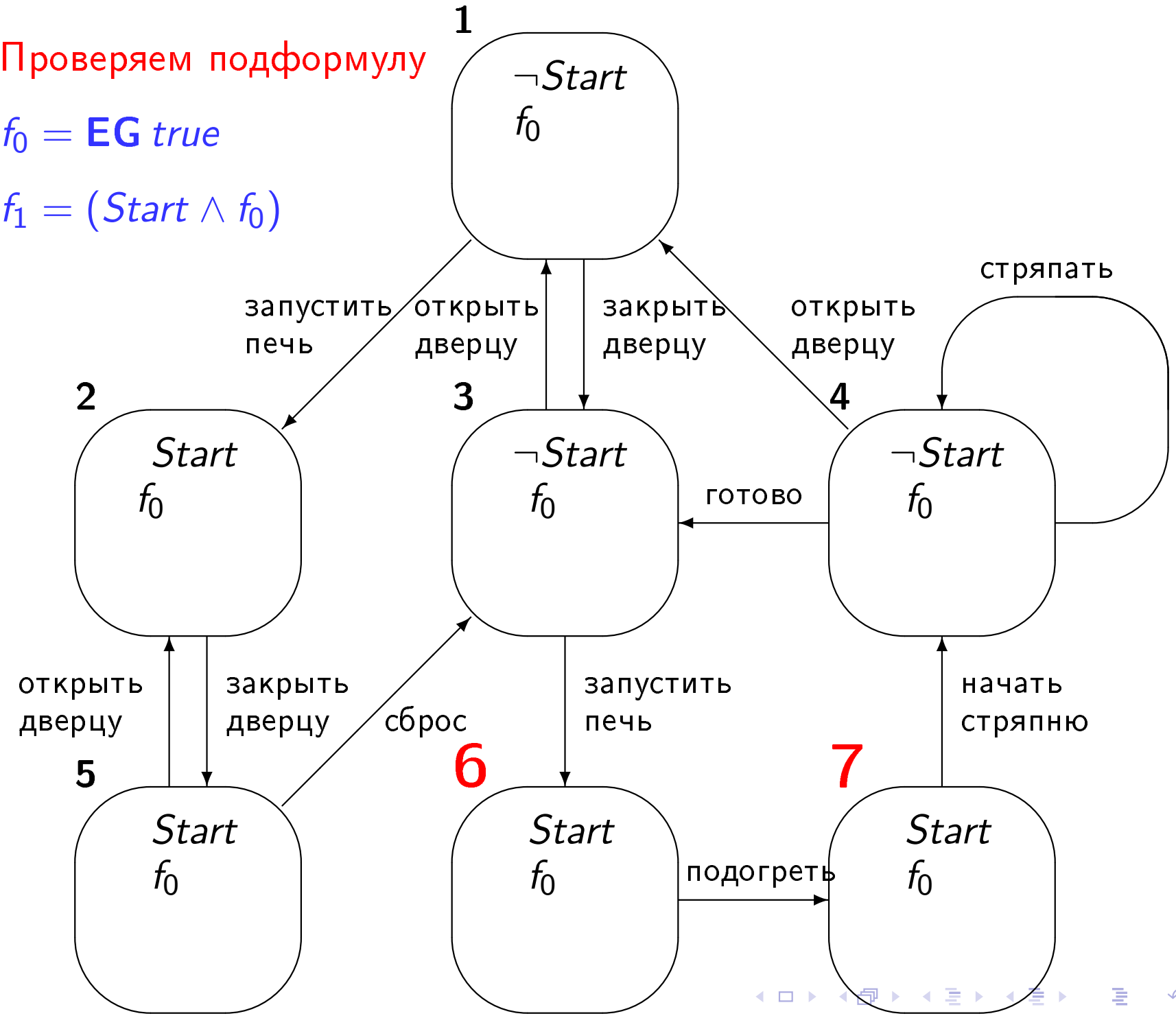
$f_0 = \mathbf{EG}\ true$



Проверяем подформулу

$f_0 = \mathbf{EG}\ true$

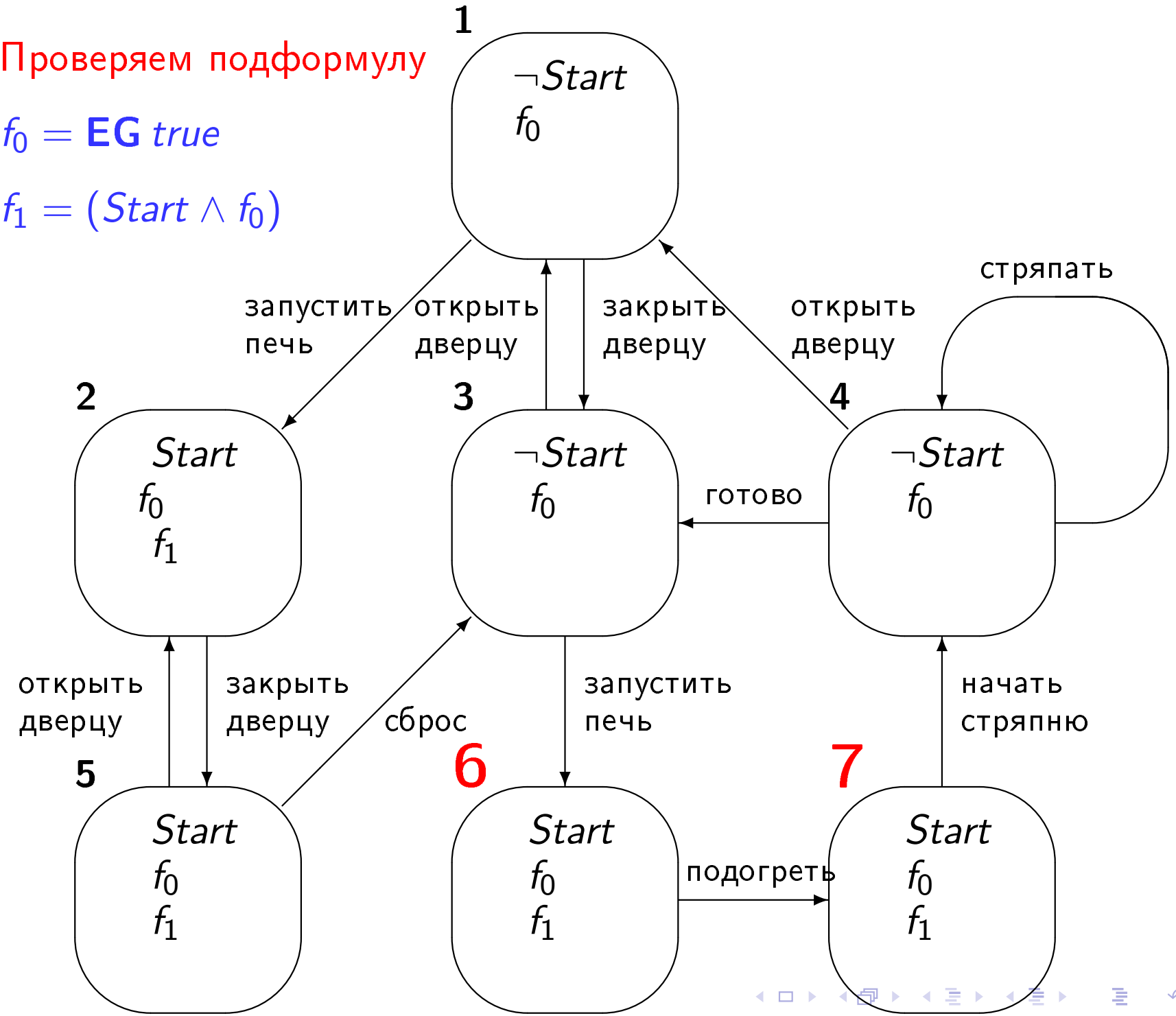
$f_1 = (Start \wedge f_0)$



Проверяем подформулу

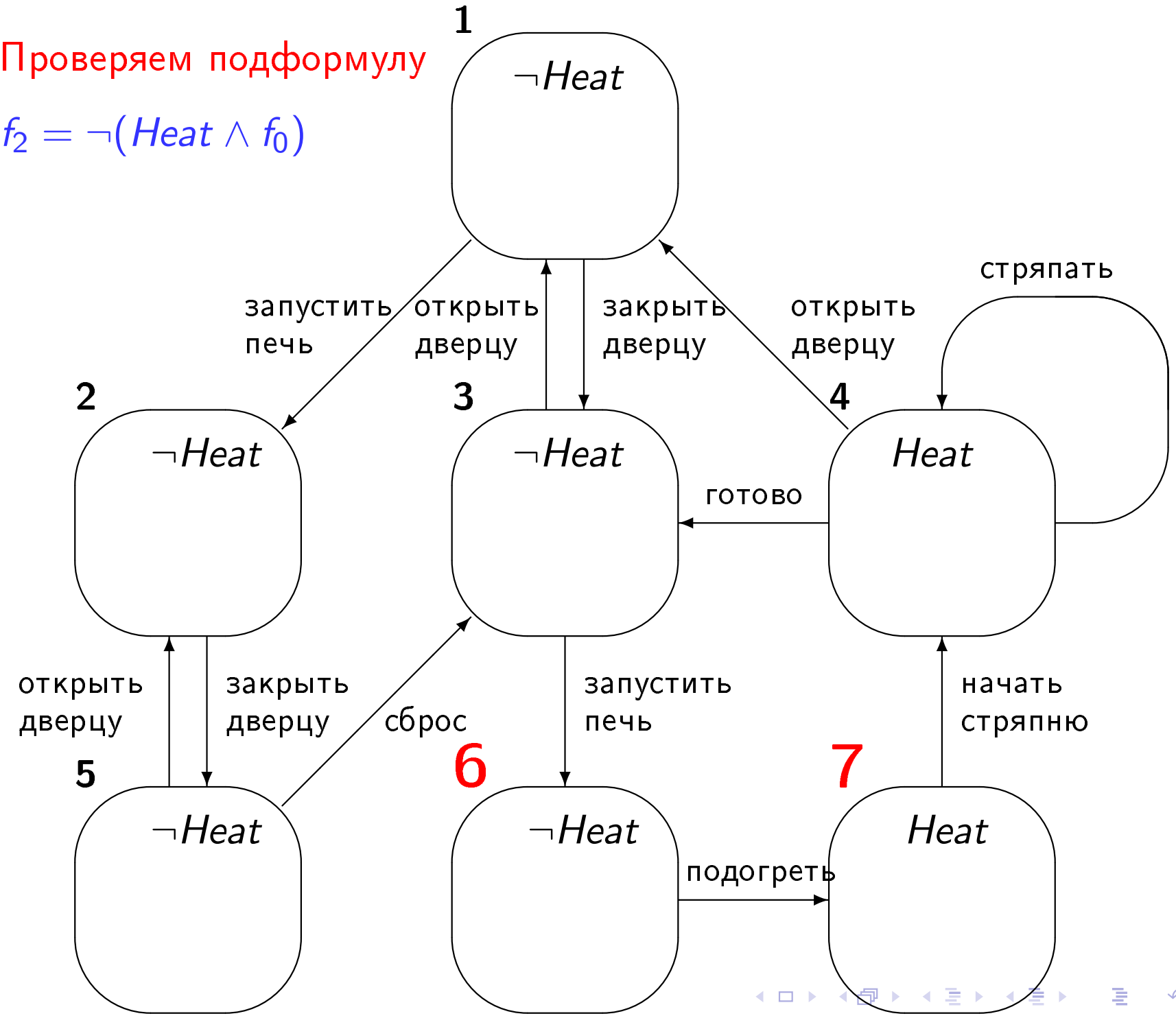
$f_0 = \mathbf{EG}\ true$

$f_1 = (Start \wedge f_0)$



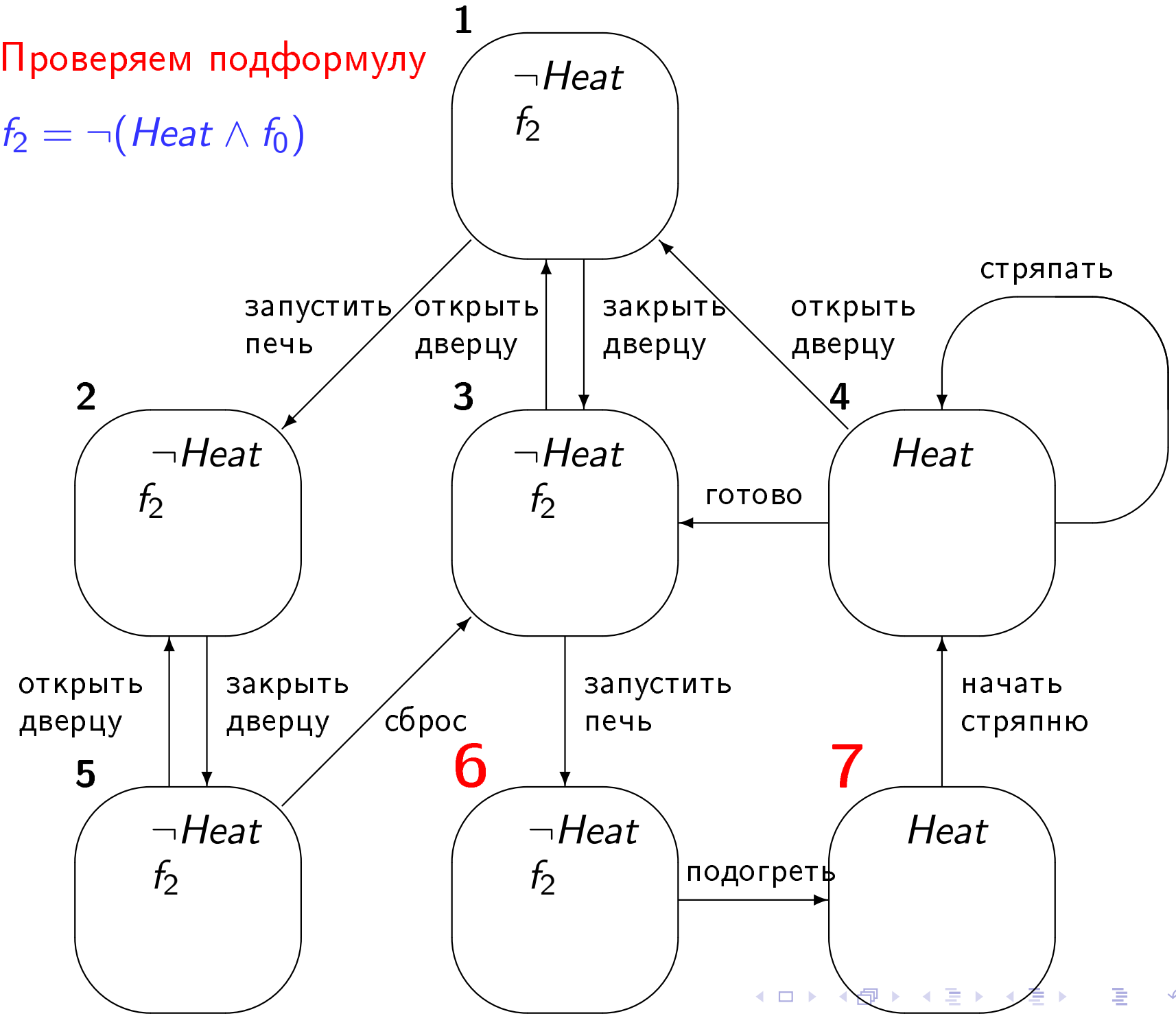
Проверяем подформулу

$f_2 = \neg(Heat \wedge f_0)$



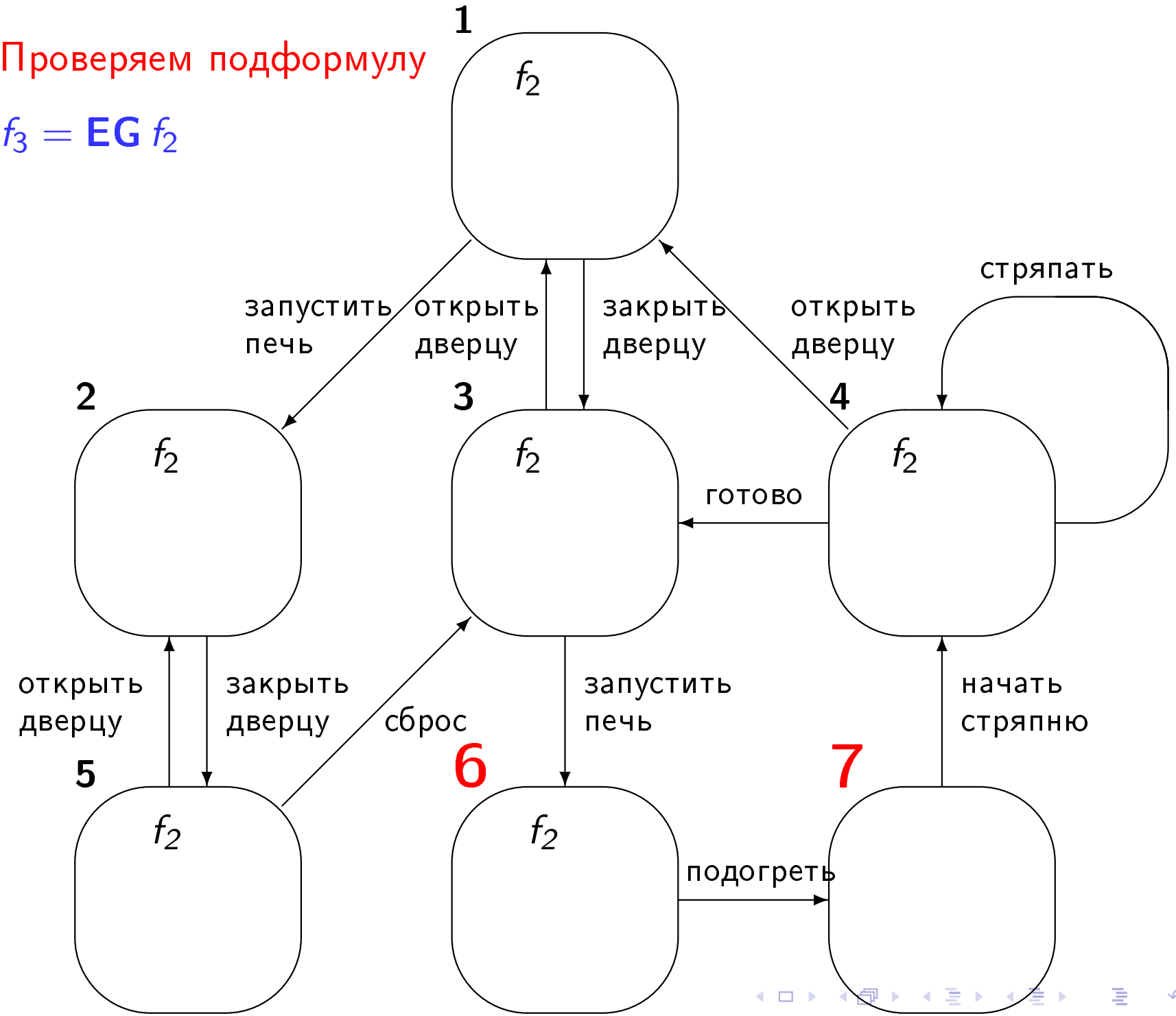
Проверяем подформулу

$f_2 = \neg(Heat \wedge f_0)$



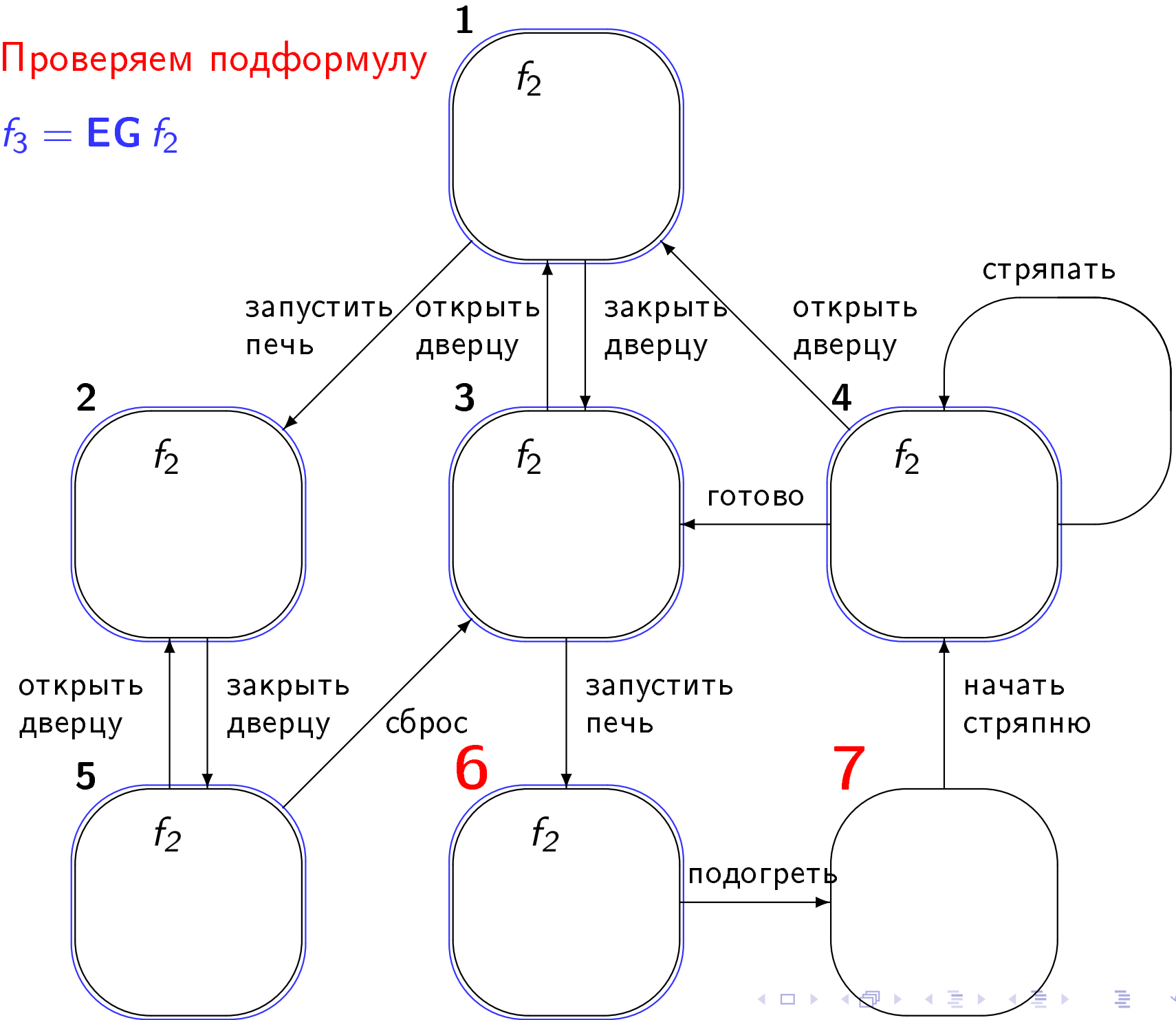
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



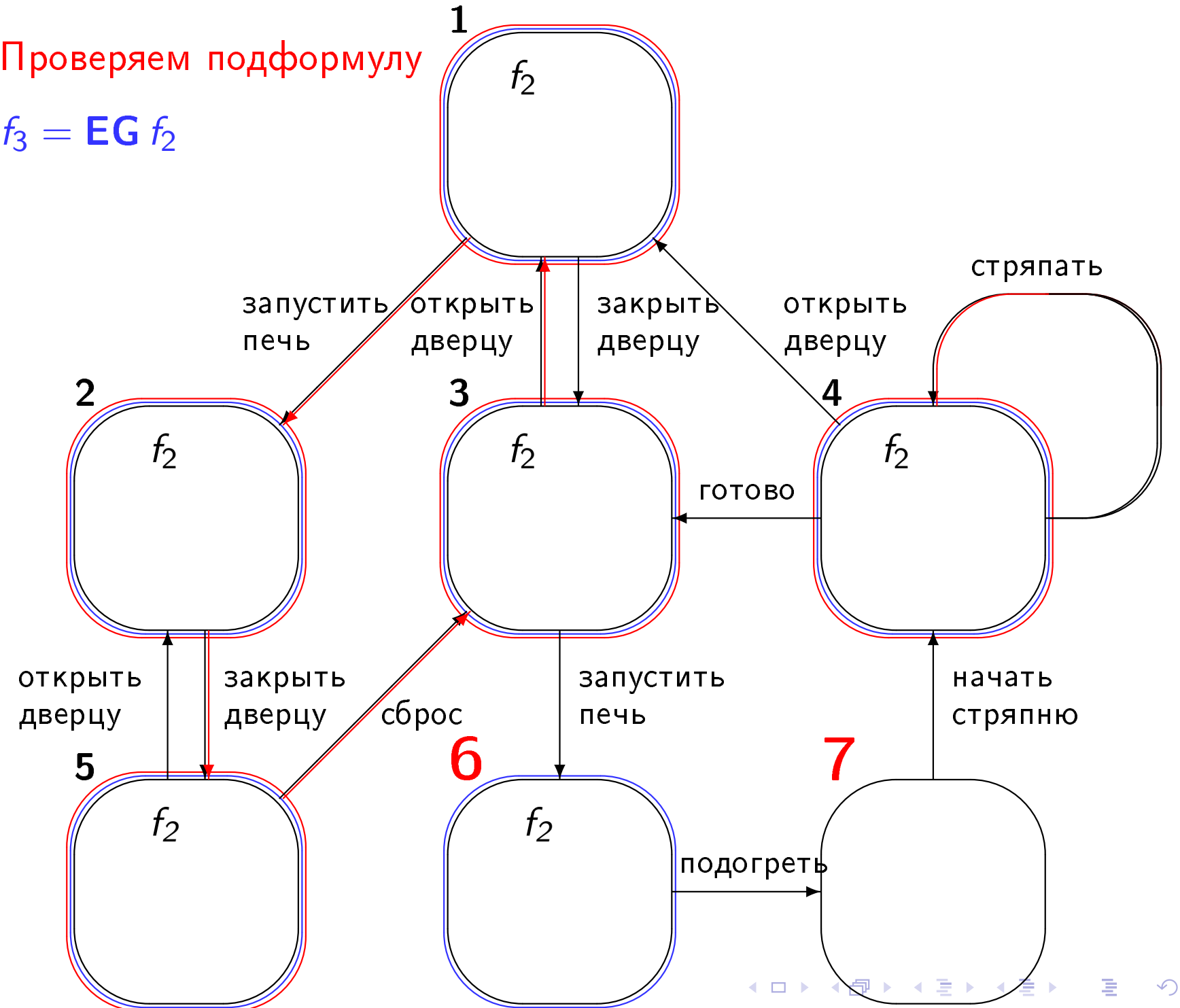
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



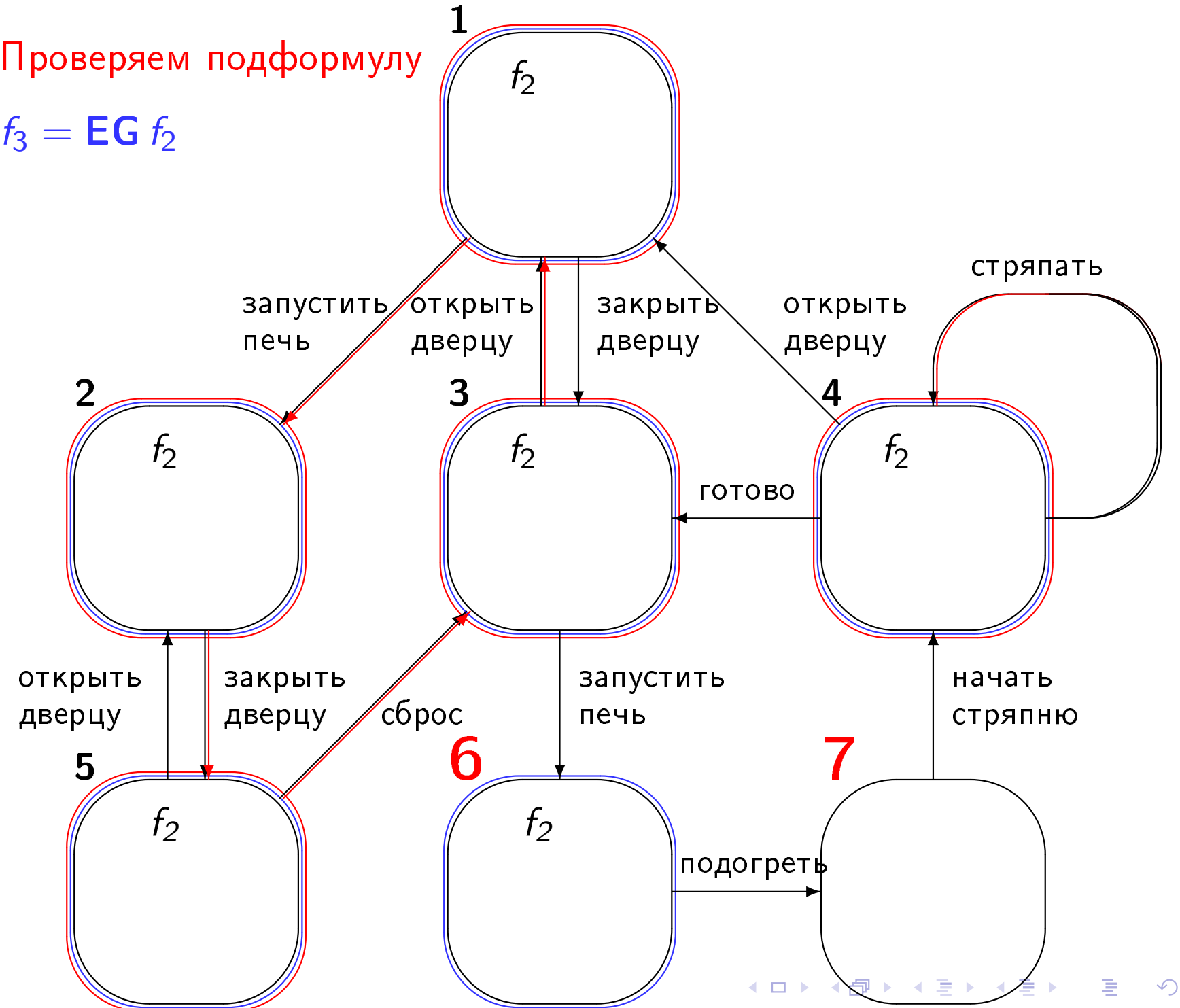
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



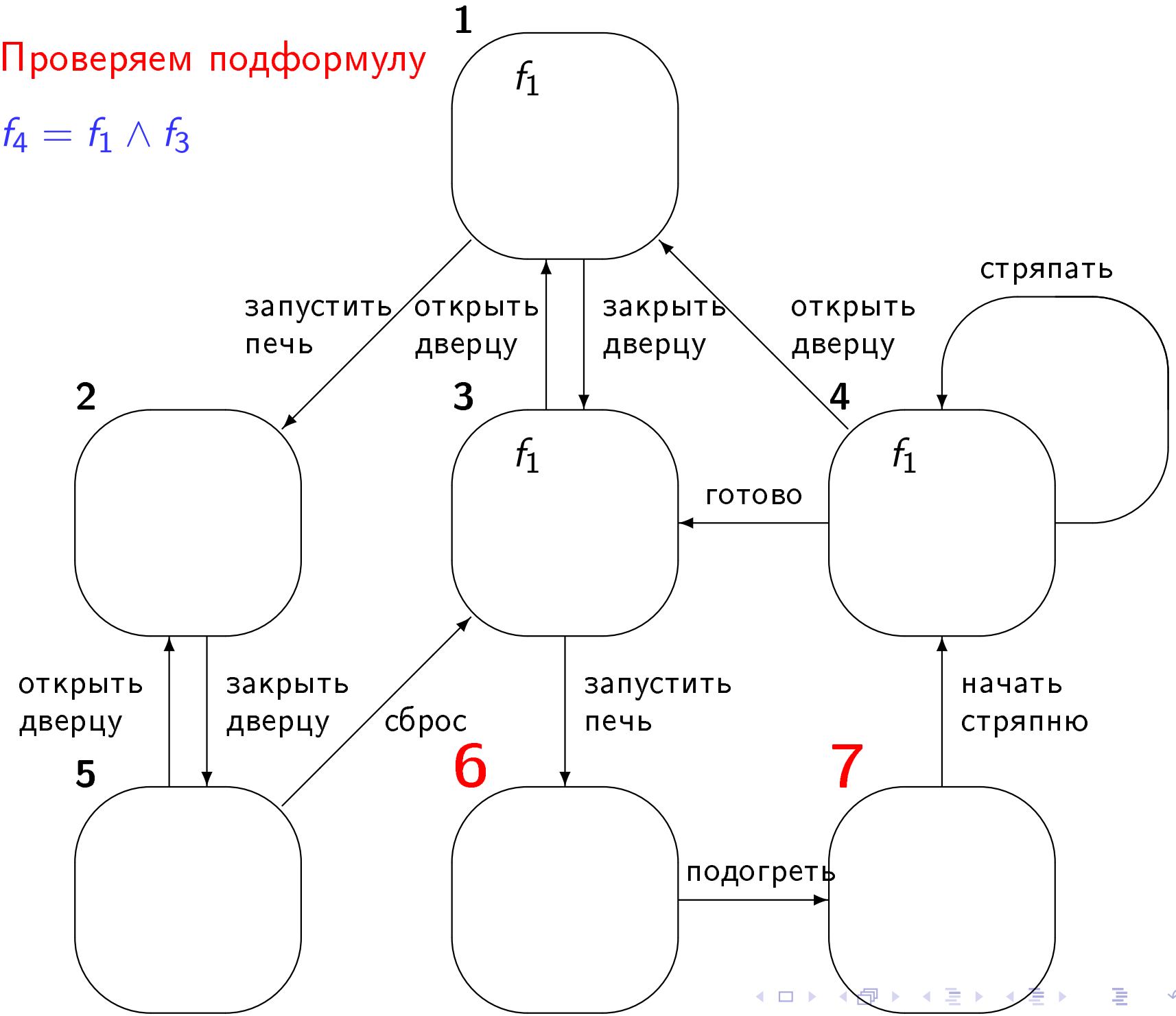
Проверяем подформулу

$f_3 = \mathbf{EG}\, f_2$



Проверяем подформулу

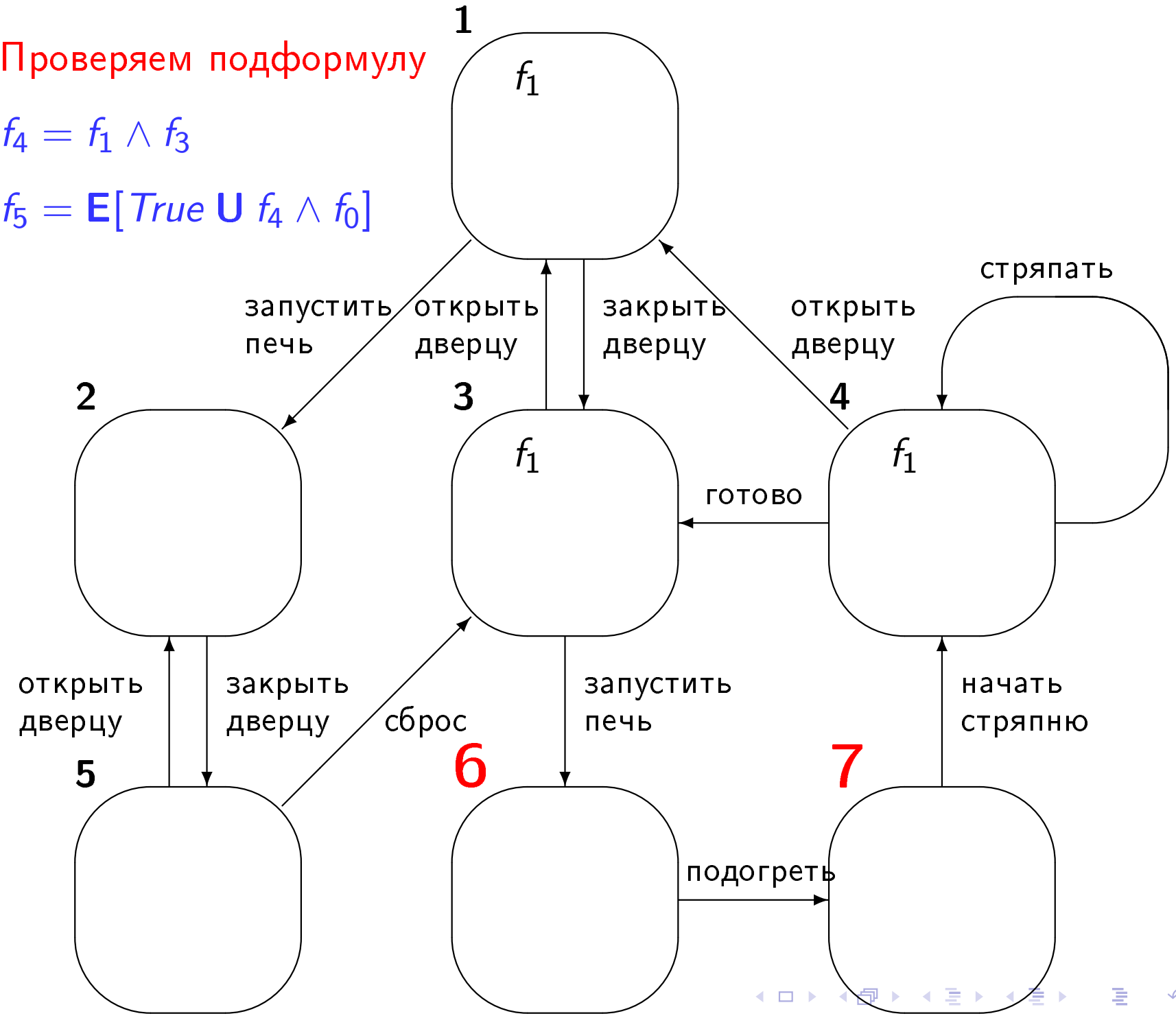
$f_4 = f_1 \wedge f_3$



Проверяем подформулу

$f_4 = f_1 \wedge f_3$

$f_5 = \mathbf{E}[True\ \mathbf{U}\ f_4 \wedge f_0]$

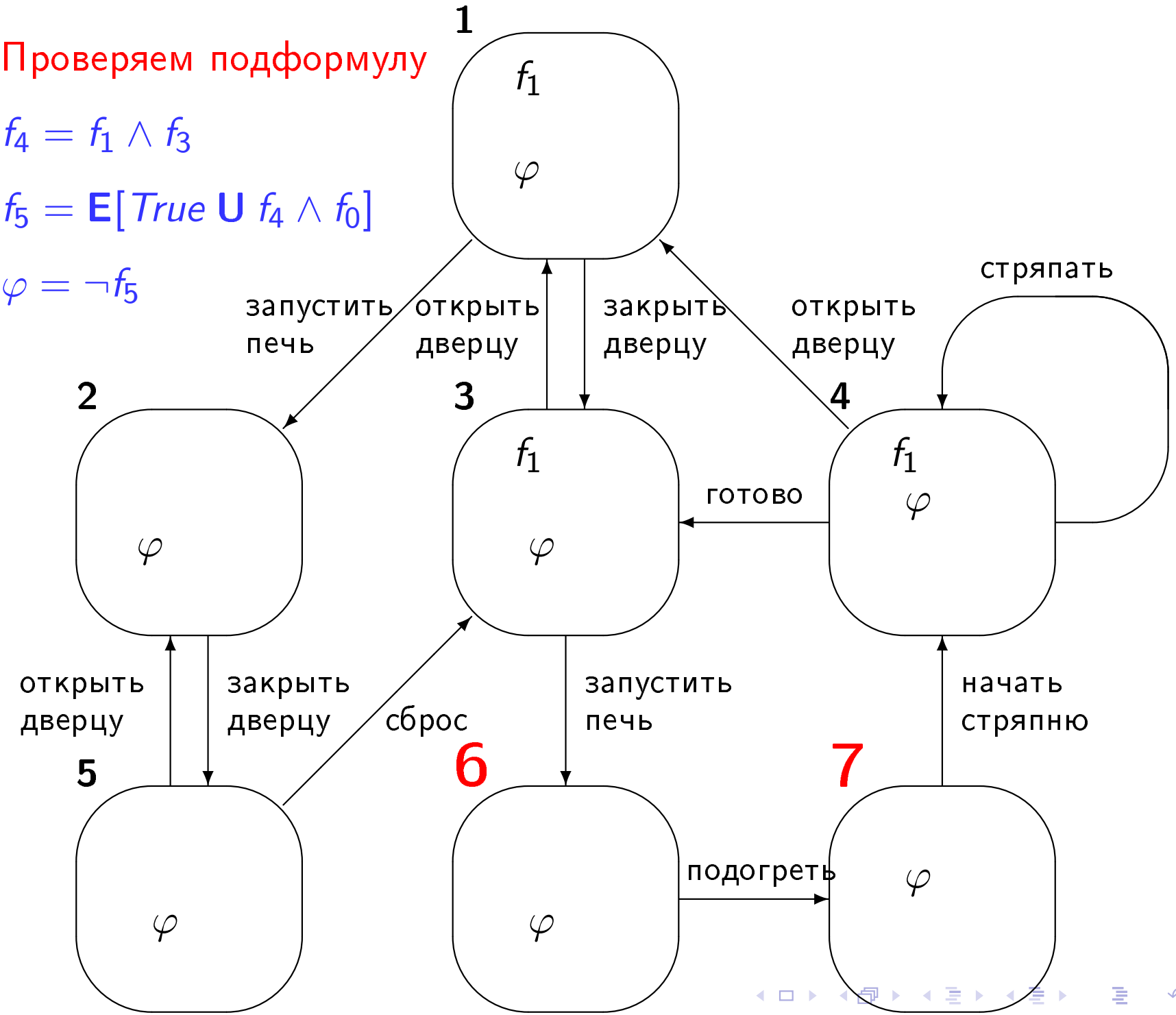


Проверяем подформулу

$f_4 = f_1 \wedge f_3$

$f_5 = \mathbf{E}[True\ \mathbf{U}\ f_4 \wedge f_0]$

$\varphi = \neg f_5$



Model checking для CTL в ограничениях справедливости

Таким образом,

$$S(\neg \mathbf{E}[\mathit{True}\ \mathbf{U}\ (\mathit{Start} \wedge \mathbf{EG}\ \neg \mathit{Heat})]) = \{1, 2, 3, 4, 5, 6, 7\}\ .$$

А это означает, что $M, 1 \models_F \mathbf{AG}(\mathit{Start} \rightarrow \mathbf{AF}\ \mathit{Heat})$, т.е. программа микроволновой печи удовлетворяет спецификации правильного поведения при заданных ограничениях справедливости.

Табличный алгоритм model checking для CTL

Достоинства предложенного алгоритма:

1. очень простой: все сводится к решению хорошо известных задач теории графов (проблемы достижимости вершин, выделения сильно связных компонент);
2. очень быстрый: время вычисления пропорционально размеру входных данных.

Табличный алгоритм model checking для CTL

Достоинства предложенного алгоритма:

1. очень простой: все сводится к решению хорошо известных задач теории графов (проблемы достижимости вершин, выделения сильно связных компонент);
2. очень быстрый: время вычисления пропорционально размеру входных данных.

Недостатки предложенного алгоритма:

1. очень медленный: время вычисления пропорционально числу состояний в модели;
2. очень затратный по объему памяти: необходимо хранить список **всех** состояний модели.

Если система состоит из 16 процессов, в которых используются 16 булевых переменных, то число состояний модели

$$2^{256}$$

Табличный алгоритм model checking для CTL

Как же работать с такими моделями?

Табличный алгоритм model checking для CTL

Как же работать с такими моделями?

Нужно проводить вычисления не с отдельными состояниями модели, а с целыми множествами состояний **одновременно**!

Если у нас есть два множества A и B, то время вычисления операция $A \cup B$ пропорционально размеру этих множеств, если множества представлены в виде списков, массивов и пр.

Но если множества X представлены короткими описаниями $description(X)$, то время вычисления $A \cup B$ может оказаться пропорциональным **размеру описаний** этих множеств, а не размеру самих множеств.

Эта идея открывает новый вид алгоритмов решения дискретных задач большой размерности — символьные алгоритмы.

Но для этого нам понадобятся средства символьного описания множеств двоичных векторов.

Двоичные разрешающие диаграммы

Одна из наиболее удобных математических структур для представления размеченных систем переходов с конечным числом состояний — это **двоичные разрешающие диаграммы (Binary Decision Diagrams, BDDs)**.

1) Вначале обсудим, как использовать двоичные разрешающие диаграммы для представления булевых функций.

2) Далее покажем, как эффективно выполнять различные логические операции, используя BDDs.

3) В заключение поясним, как при помощи BDDs компактно представлять модели Крипке.

Основные определения

Упорядоченные двоичные разрешающие диаграммы (или, сокращенно, OBDD — ordered binary decision diagrams) — это каноническая форма представления булевых функций.

Они значительно более компактны, нежели традиционные нормальные формы, такие как ДНФ или КНФ.

Кроме того, с ними можно работать очень эффективно.

Поэтому OBDDs нашли широкое применение во многих приложениях, относящихся к автоматическому проектированию, символьному имитационному моделированию, верификации комбинационных логических схем.

Основные определения

Вначале рассмотрим двоичные разрешающие деревья.

Двоичное разрешающее дерево — это корневое ориентированное дерево, вершины которого разбиты на два класса — терминальные вершины и нетерминальные вершины.

Каждая нетерминальная вершина v помечена переменной $var(v)$ и имеет две вершины–последователя: $low(v)$, которая соответствует случаю, когда переменная $var(v)$ равна 0, и $high(v)$, которая соответствует случаю, когда переменная $var(v)$ равна 1.

Каждая из терминальных вершин v имеет пометку $value(v)$, равную 0 или 1.

Основные определения

Вначале рассмотрим двоичные разрешающие деревья.

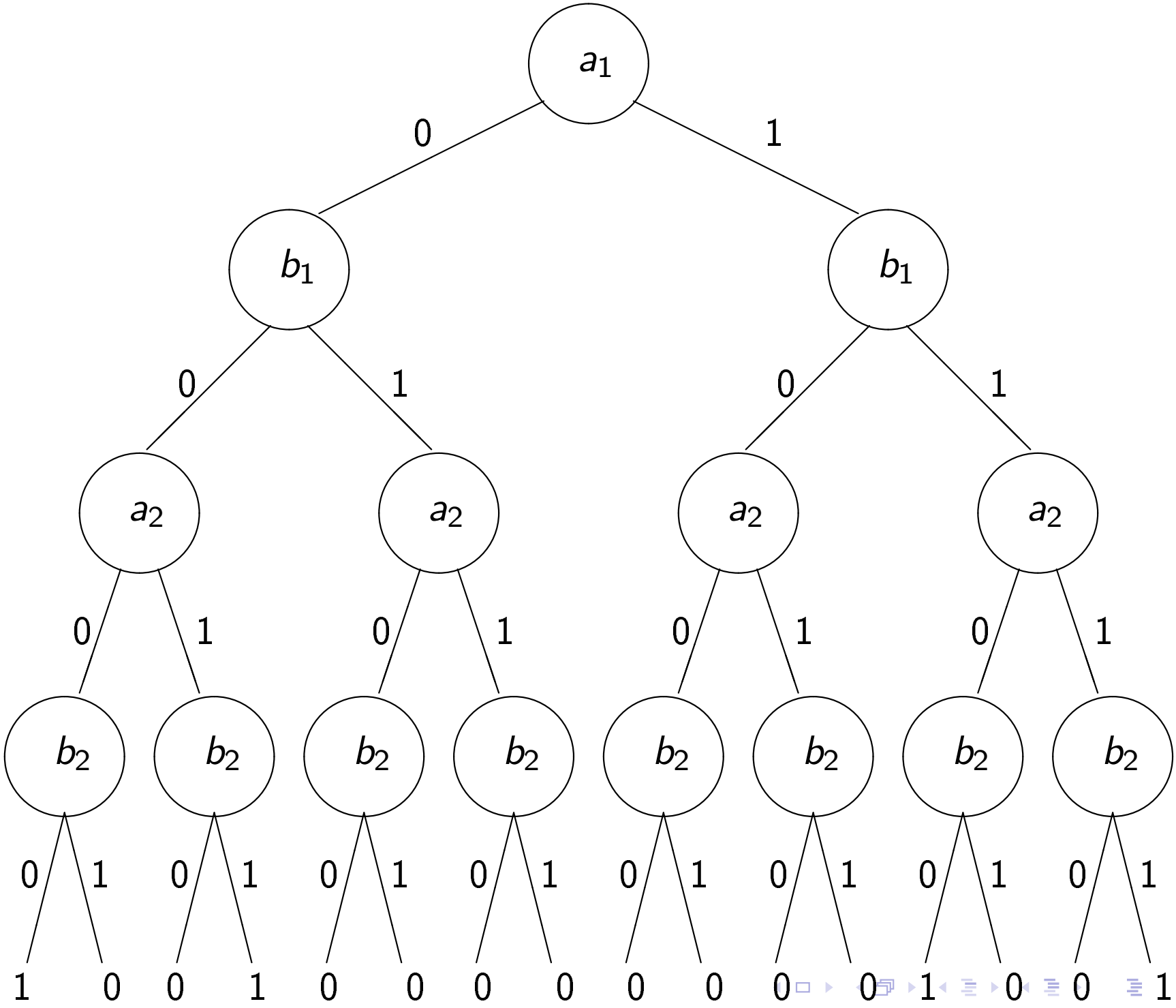
Двоичное разрешающее дерево — это корневое ориентированное дерево, вершины которого разбиты на два класса — терминальные вершины и нетерминальные вершины.

Каждая нетерминальная вершина v помечена переменной $var(v)$ и имеет две вершины–последователя: $low(v)$, которая соответствует случаю, когда переменная $var(v)$ равна 0, и $high(v)$, которая соответствует случаю, когда переменная $var(v)$ равна 1.

Каждая из терминальных вершин v имеет пометку $value(v)$, равную 0 или 1.

Рассмотрим двоичное разрешающее дерево для двухбитового компаратора, который описывается формулой

$$f(a_1, a_2, b_1, b_2) = (a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2).$$


a_1
0
1
b_1
b_1
0
1
0
1
a_2
a_2
a_2
a_2
0
1
0
1
0
1
0
1
b_2
b_2
b_2
b_2
b_2
b_2
b_2
b_2
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
1
0
0
1
0
0
0
0
0
0
0
0
1
0
0
1

Основные определения

Чтобы выяснить, какое значение принимает формула при заданном наборе значений переменных, нужно спустится по дереву из корня до терминальной вершины.

Если переменная $var(v)$ имеет значение 0 , то следующей вершиной на пути из корня в терминальную вершину будет $low(x)$.

Если же $var(v)$ имеет значение 1 , то последующей вершиной на пути будет $high(x)$.

То значение, которым помечена достигнутая терминальная вершина, и будет значением функции на этом наборе.

Основные определения

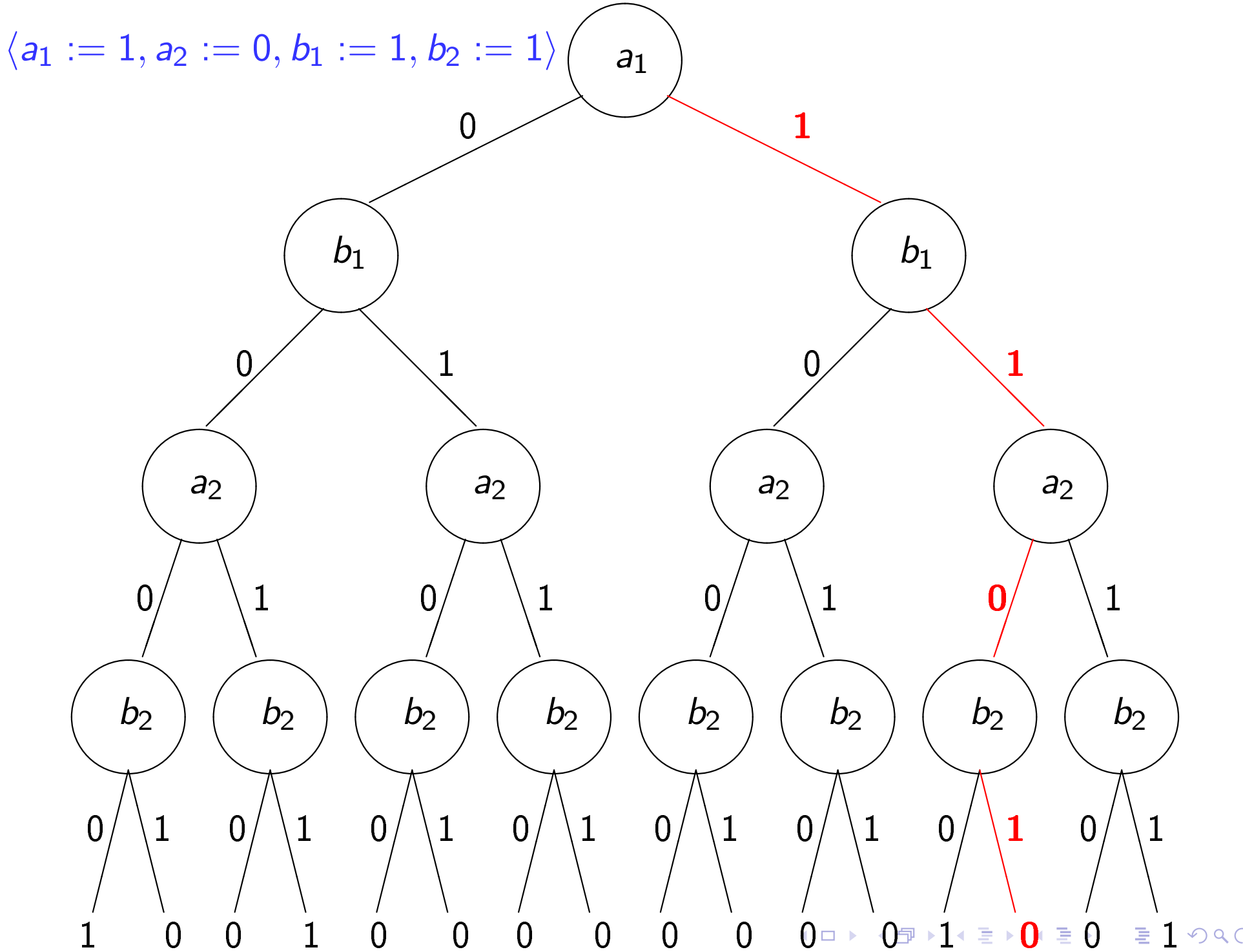
Чтобы выяснить, какое значение принимает формула при заданном наборе значений переменных, нужно спустится по дереву из корня до терминальной вершины.

Если переменная $var(v)$ имеет значение 0 , то следующей вершиной на пути из корня в терминальную вершину будет $low(x)$.

Если же $var(v)$ имеет значение 1 , то последующей вершиной на пути будет $high(x)$.

То значение, которым помечена достигнутая терминальная вершина, и будет значением функции на этом наборе.

Например, оценка $\langle a_1 := 1, a_2 := 0, b_1 := 1, b_2 := 1 \rangle$ приводит к висячей вершине, помеченной 0 ; следовательно, при таком означивании формула ложна.


⟨a1 := 1, a2 := 0, b1 := 1, b2 := 1⟩
a1
b1
b1
a2
a2
a2
a2
b2
b2
b2
b2
b2
b2
b2
b2
0
1

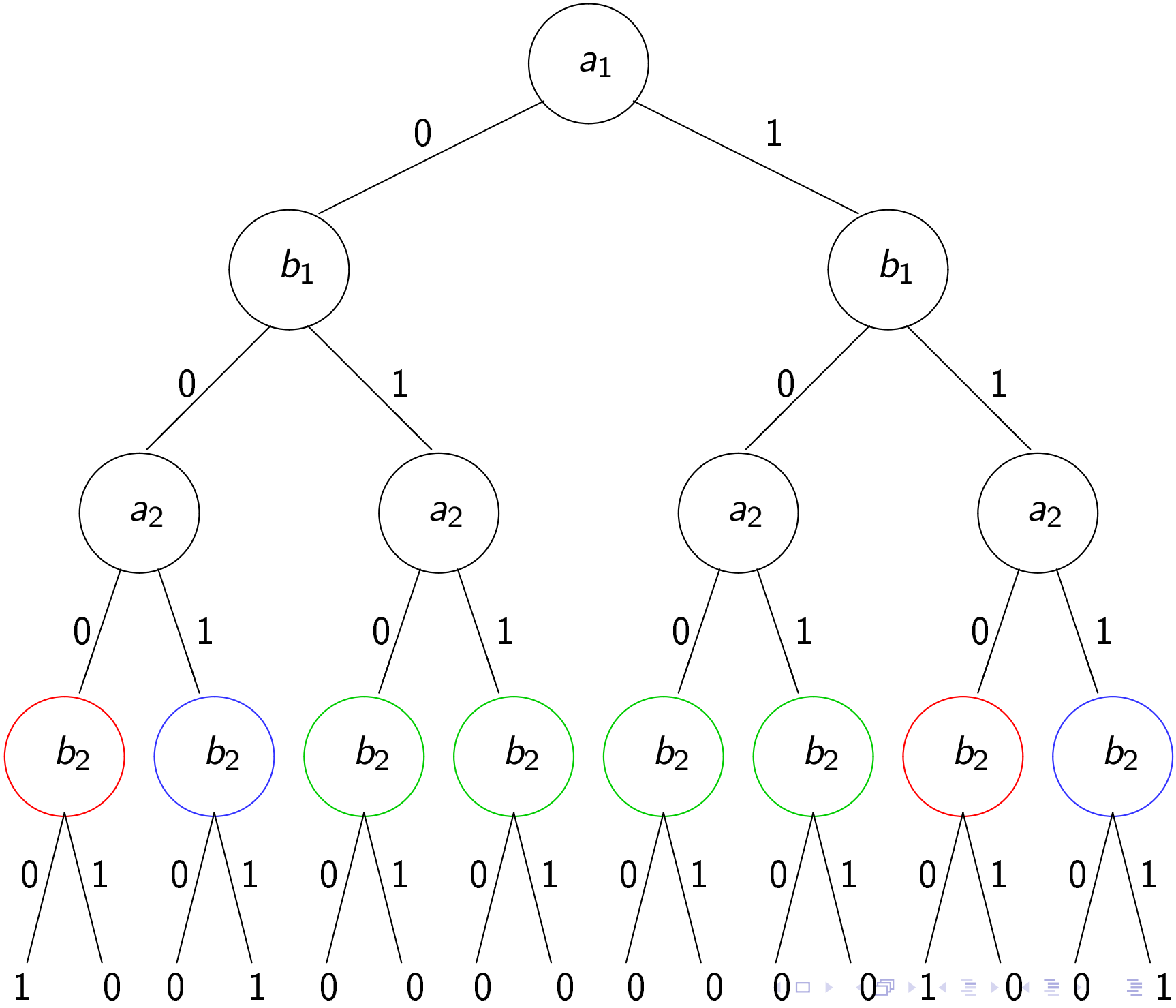
Основные определения

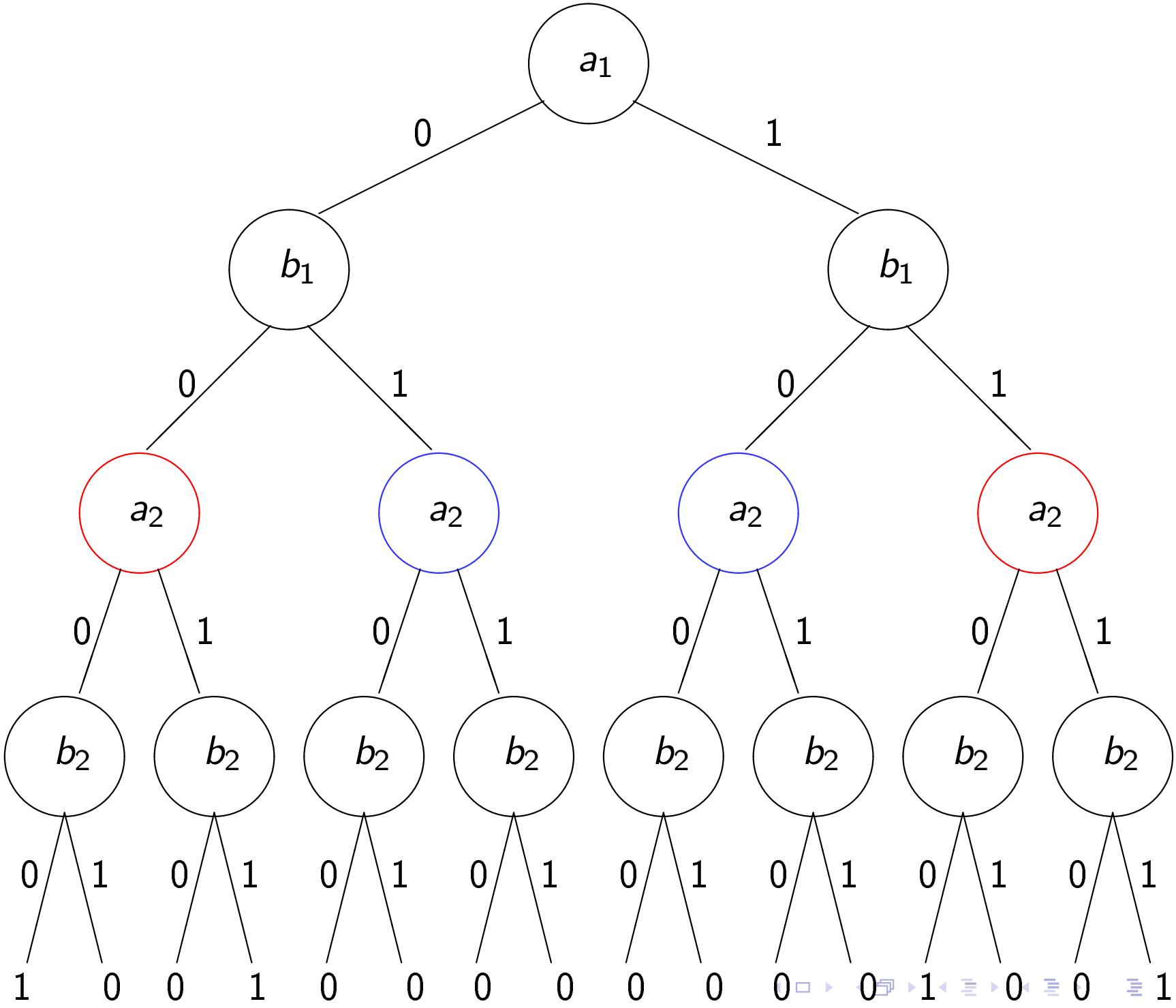
Двоичные разрешающие деревья не дают очень компактного представления булевых функций. По сути дела, они имеют точно такой же размер, как и таблица истинности.

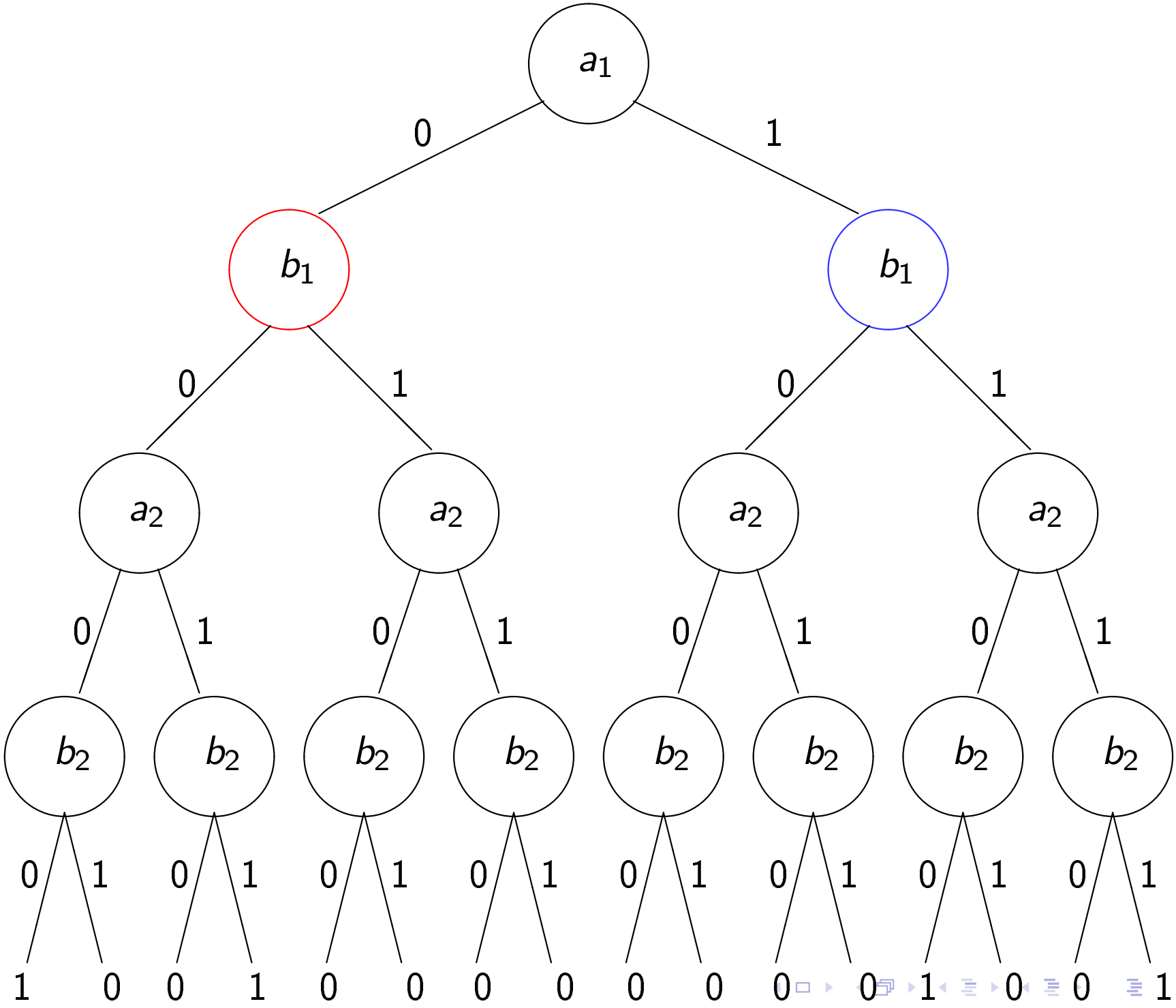
Однако такие деревья обычно бывают чрезвычайно избыточными. Например, дерево двухбитового компаратора содержит восемь поддеревьев, корни которых помечены b_2 , но только три из них попарно различны.

За счет этого мы можем добиться более компактного представления для булевых функций, как только склеим все изоморфные поддеревья.

В результате получится ориентированный ациклический граф, который и называется двоичной разрешающей диаграммой .


a_1
0
1
b_1
b_1
0
1
0
1
a_2
a_2
a_2
a_2
0
1
0
1
0
1
0
1
b_2
b_2
b_2
b_2
b_2
b_2
b_2
b_2
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
1
0
0
1
0
0
0
0
0
0
0
0
1
0
0
1


a_1
0
1
b_1
b_1
0
1
0
1
a_2
a_2
a_2
a_2
0
1
0
1
0
1
0
1
b_2
b_2
b_2
b_2
b_2
b_2
b_2
b_2
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
1
0
0
1
0
0
0
0
0
0
0
0
1
0
0
1


a_1
0
1
b_1
b_1
0
1
0
1
a_2
a_2
a_2
a_2
0
1
0
1
0
1
0
1
b_2
b_2
b_2
b_2
b_2
b_2
b_2
b_2
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
1
0
0
1
0
0
0
0
0
0
0
0
0
1
0
1

Основные определения

Двоичная разрешающая диаграмма — это корневой ориентированный ациклический граф, вершины которого разбиты на два класса — терминальные вершины и нетерминальные вершины.

Как и в случае двоичных разрешающих деревьев, каждая нетерминальная вершина v помечена переменной $var(v)$ и имеет две вершины–последователя: $low(v)$ и $high(v)$. Каждая терминальная вершина помечена либо 0, либо 1.

Основные определения

Всякая двоичная разрешающая диаграмма B с корнем в вершине v определяет булеву функцию $f_v(x_1, \ldots, x_n)$ следующего вида:

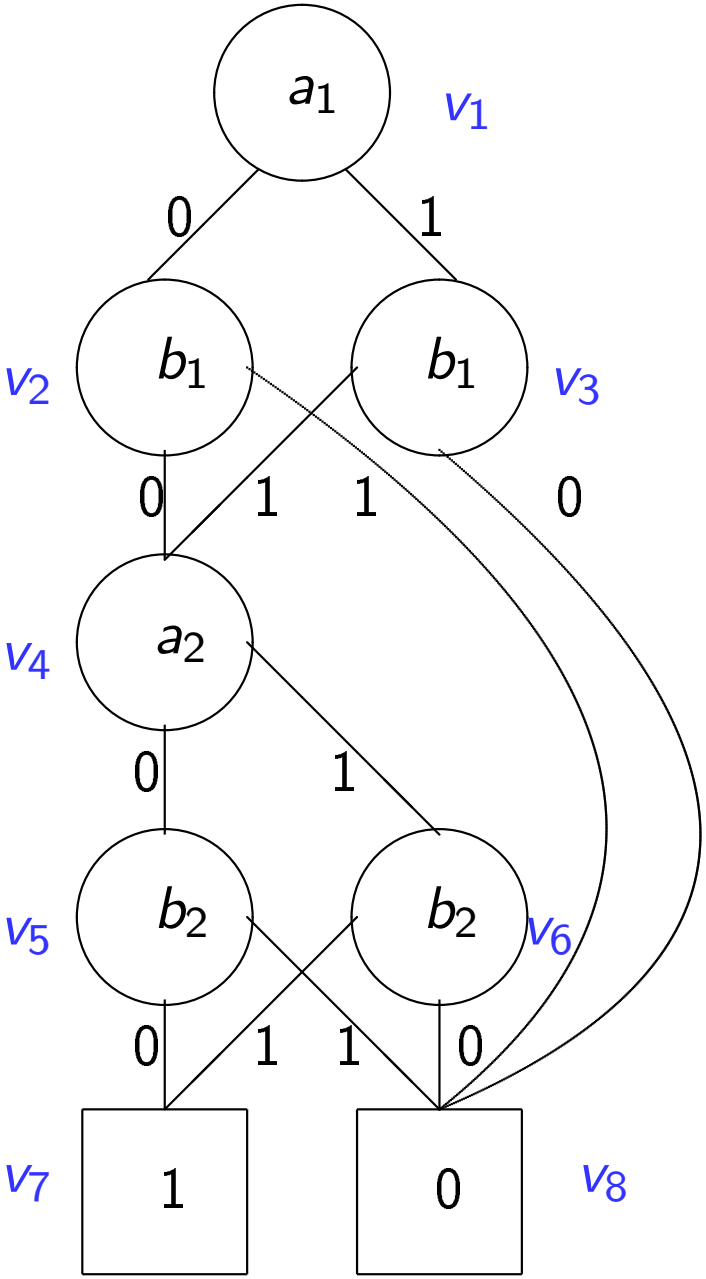
1. В случае, когда v — терминальная вершина, мы считаем, что

   a). если $value(v) = 1$, то $f_v(x_1, \ldots, x_n) = 1$,
   b). если $value(v) = 0$, то $f_v(x_1, \ldots, x_n) = 0$.

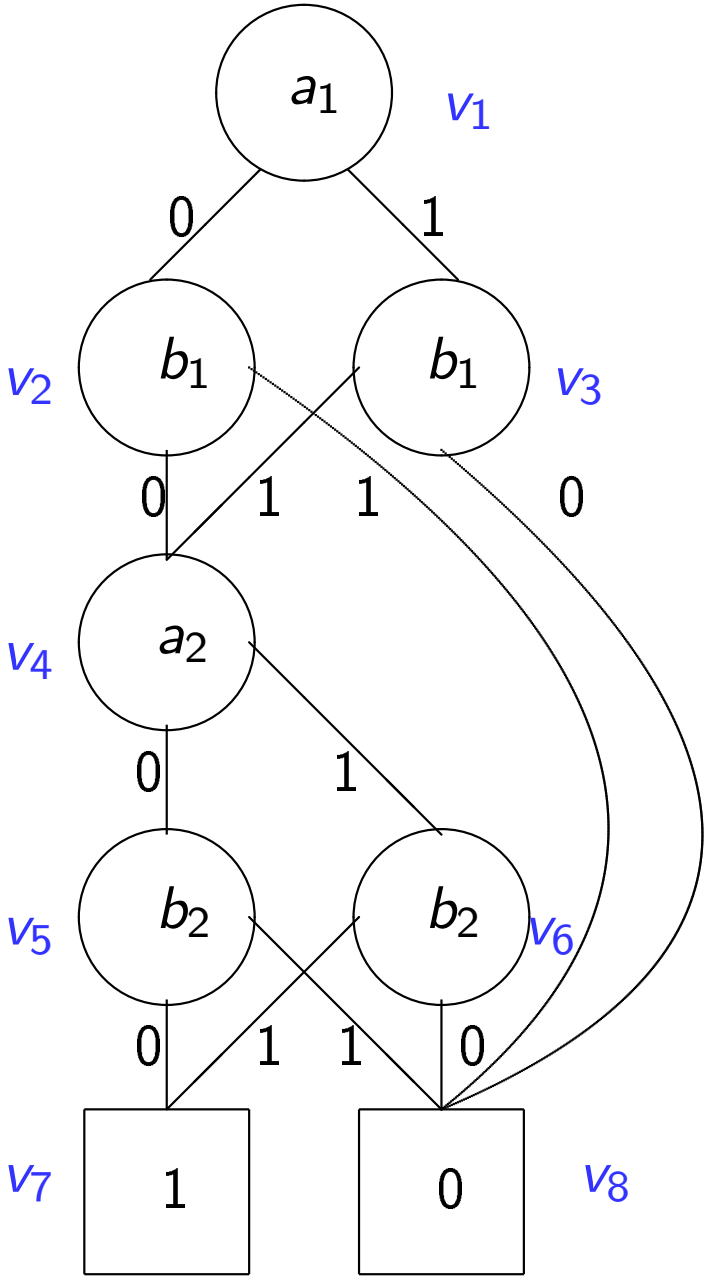
2. В случае, когда v — нетерминальная вершина и $var(v) = x_i$ мы имеем

$$f_v(x_1, \ldots, x_n) = (\neg x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee (x_i \wedge f_{high(v)}(x_1, \ldots, x_n)).$$

Основные определения

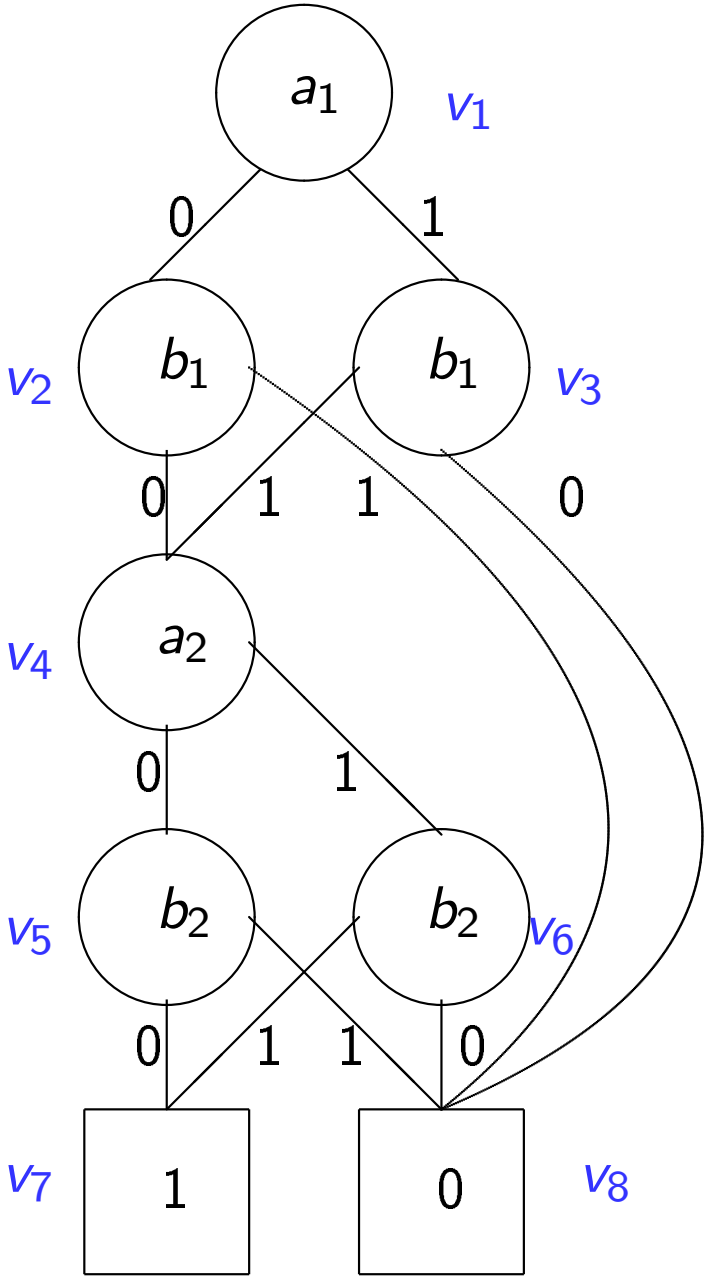


Основные определения



$$f_{v_7} = 1 \quad f_{v_8} = 0$$

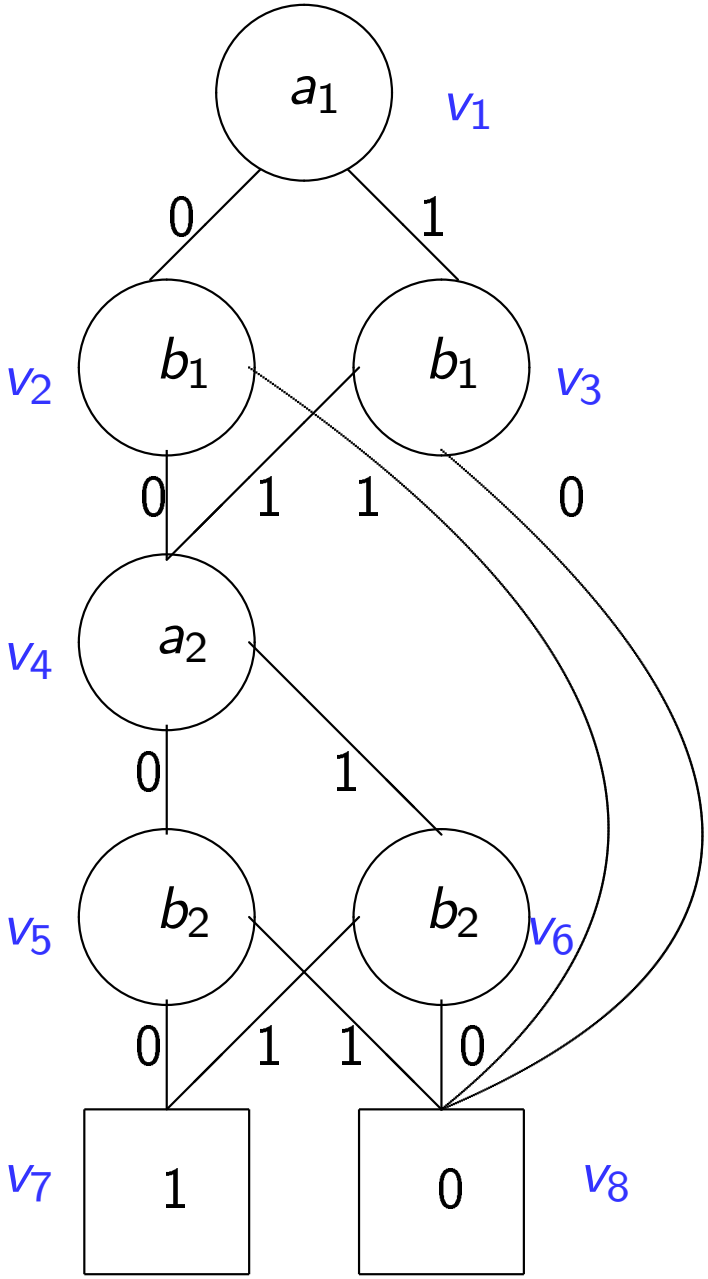
Основные определения



$$f_{v_5} = \neg b_2 \quad f_{v_5} = b_2$$

$$f_{v_7} = 1 \quad f_{v_8} = 0$$

Основные определения

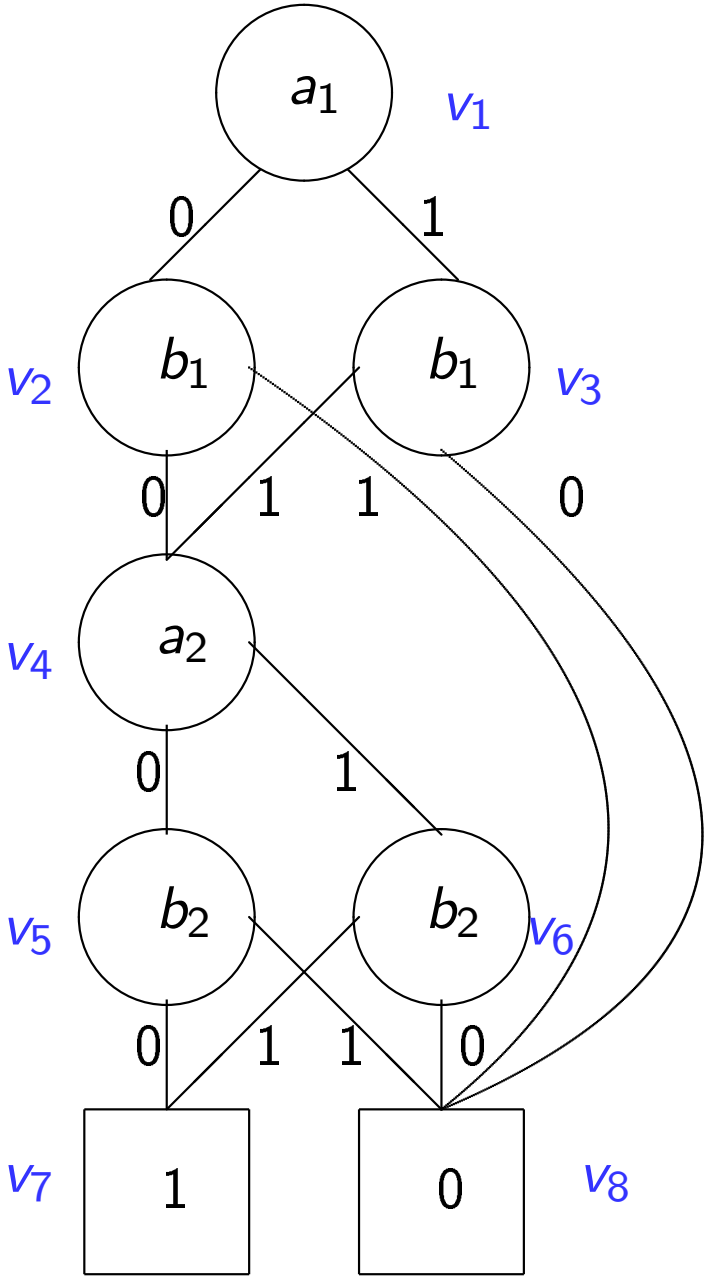


$$f_{v_4} = (\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2)$$

$$f_{v_5} = \neg b_2 \quad f_{v_5} = b_2$$

$$f_{v_7} = 1 \quad f_{v_8} = 0$$

Основные определения

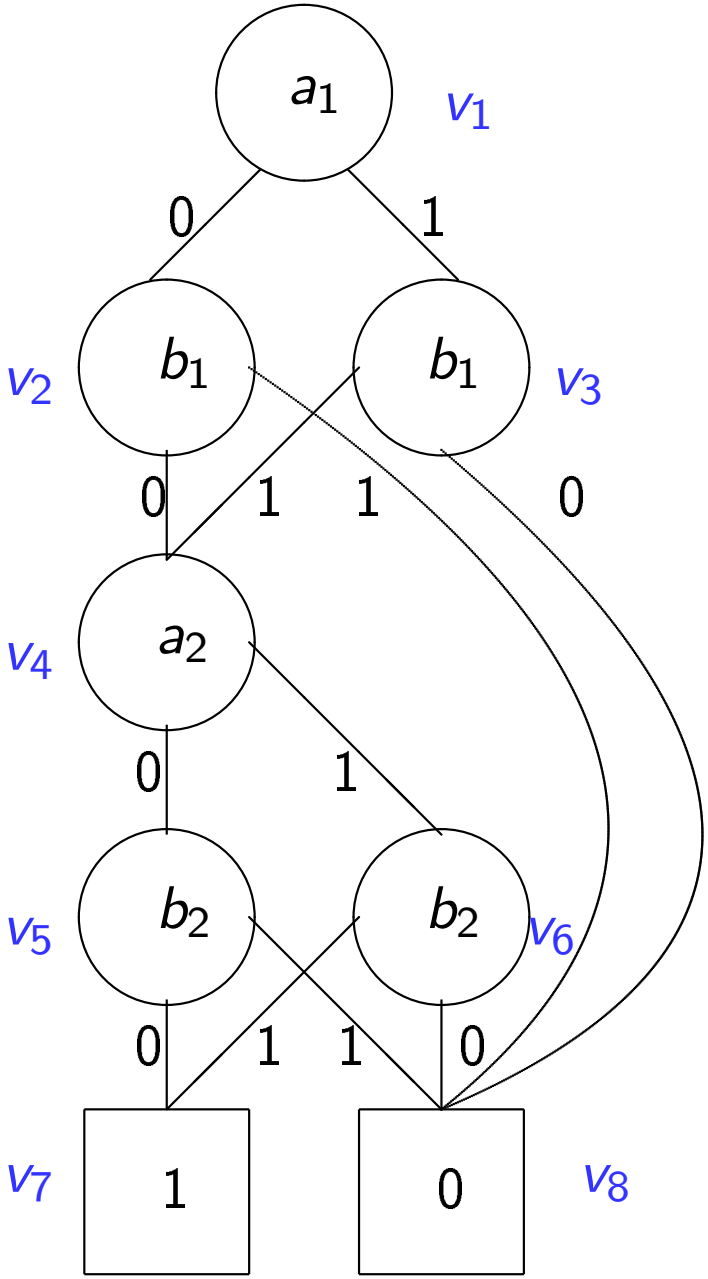


$$f_{v_4} = (\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2)$$
$$= (a_2 \leftrightarrow b_2)$$

$$f_{v_5} = \neg b_2 \quad f_{v_5} = b_2$$

$$f_{v_7} = 1 \quad f_{v_8} = 0$$

Основные определения



$$f_{v_2} = \neg b_1 \wedge (a_2 \leftrightarrow b_2)$$

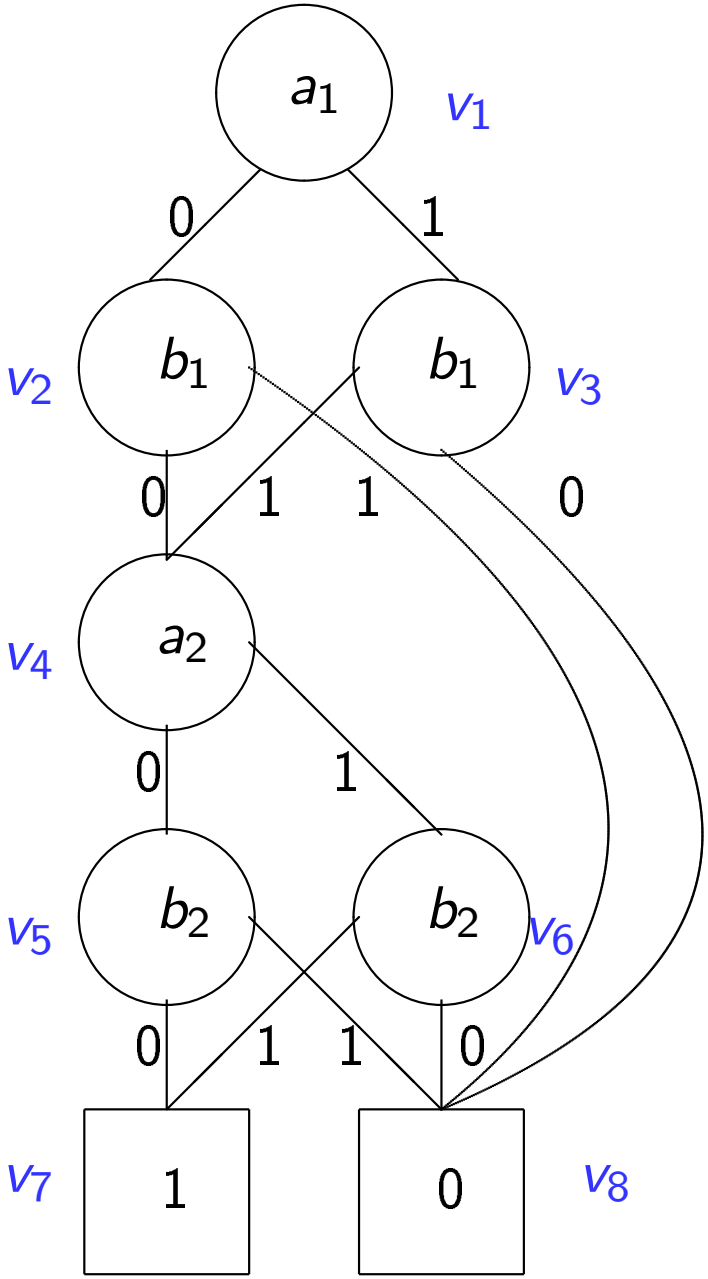
$$f_{v_3} = b_1 \wedge (a_2 \leftrightarrow b_2)$$

$$f_{v_4} = (\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2) = (a_2 \leftrightarrow b_2)$$

$$f_{v_5} = \neg b_2 \quad f_{v_5} = b_2$$

$$f_{v_7} = 1 \quad f_{v_8} = 0$$

Основные определения



$$f_{v_1} = (\neg a_1 \wedge f_{v_2}) \vee (a_1 \wedge f_{v_3})$$

$$f_{v_2} = \neg b_1 \wedge (a_2 \leftrightarrow b_2)$$

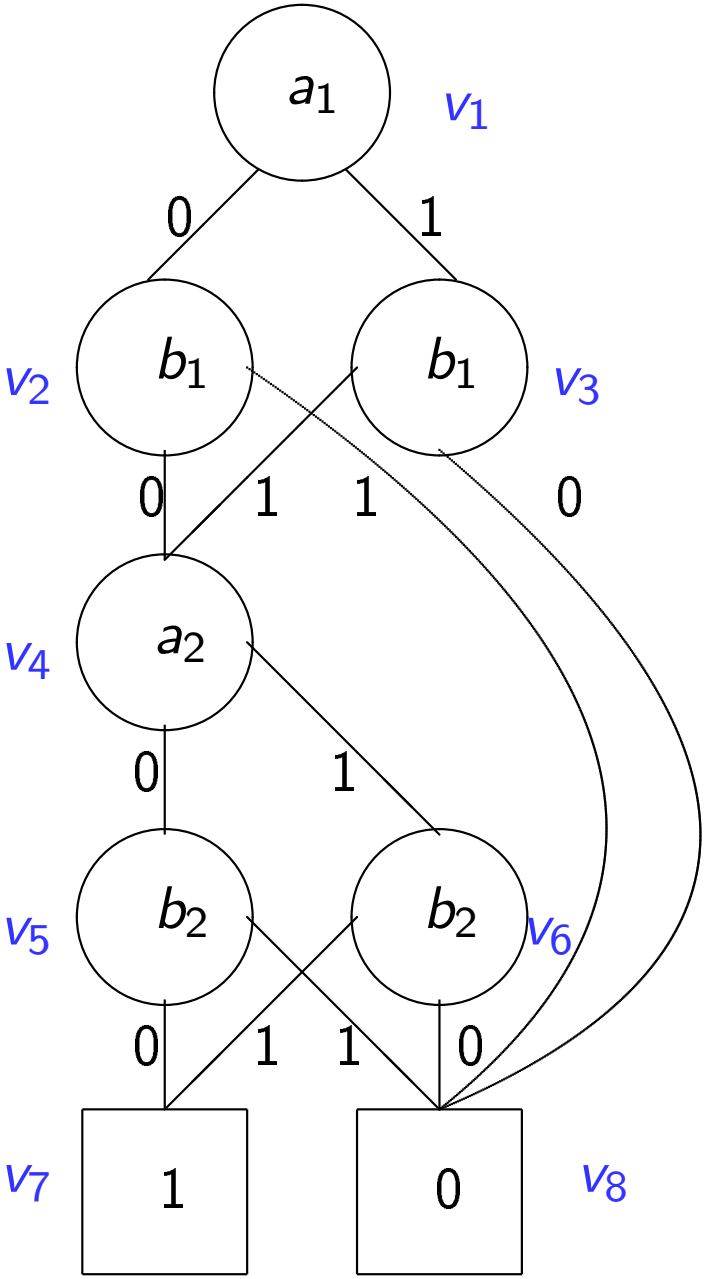
$$f_{v_3} = b_1 \wedge (a_2 \leftrightarrow b_2)$$

$$f_{v_4} = (\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2)$$
$$= (a_2 \leftrightarrow b_2)$$

$$f_{v_5} = \neg b_2 \quad f_{v_5} = b_2$$

$$f_{v_7} = 1 \quad f_{v_8} = 0$$

Основные определения



$$f_{v_1} = (\neg a_1 \wedge f_{v_2}) \vee (a_1 \wedge f_{v_3}) = (a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2)$$

$$f_{v_2} = \neg b_1 \wedge (a_2 \leftrightarrow b_2)$$

$$f_{v_3} = b_1 \wedge (a_2 \leftrightarrow b_2)$$

$$f_{v_4} = (\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2) = (a_2 \leftrightarrow b_2)$$

$$f_{v_5} = \neg b_2 \quad f_{v_5} = b_2$$

$$f_{v_7} = 1 \quad f_{v_8} = 0$$

Основные определения

Во многих приложениях важно иметь каноническое представление булевых функций, которое обладает тем свойством, что две булевы функции логически эквивалентны тогда и только тогда, когда они имеют изоморфные представления. Это свойство упрощает решение задачи сравнения двух булевых функций в итеративных алгоритмах.

Основные определения

Во многих приложениях важно иметь каноническое представление булевых функций, которое обладает тем свойством, что две булевы функции логически эквивалентны тогда и только тогда, когда они имеют изоморфные представления. Это свойство упрощает решение задачи сравнения двух булевых функций в итеративных алгоритмах.

Две BDDs считаются изоморфными, если существует такое взимно однозначное отображение h, которое сопоставляет терминальные вершины одной из них терминальным вершинам другой, а также нетерминальные вершины одной — нетерминальным вершинам другой, так что для каждой терминальной вершины v справедливо равенство $value(v) = value(h(v))$, а для каждой нетерминальной вершины v выполняются соотношения $var(v) = var(h(v))$, $h(low(v)) = low(h(v))$ и $h(high(v)) = high(h(v))$.

Построение разрешающих диаграмм

Бриан показал, как построить канонические представления булевых функций, налагая два ограничения на структуру двоичных разрешающих диаграмм.

R. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8):677–691, 1986.

Построение разрешающих диаграмм

Бриан показал, как построить канонические представления булевых функций, налагая два ограничения на структуру двоичных разрешающих диаграмм.

R. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8):677–691, 1986.

1) в каждом пути из корня в терминальную вершину переменные должны следовать в одном и том же порядке.

Построение разрешающих диаграмм

Бриан показал, как построить канонические представления булевых функций, налагая два ограничения на структуру двоичных разрешающих диаграмм.

R. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8):677–691, 1986.

1) в каждом пути из корня в терминальную вершину переменные должны следовать в одном и том же порядке.

2) в диаграмме не должно быть изоморфных поддеревьев или избыточных вершин.

Построение разрешающих диаграмм

Бриан показал, как построить канонические представления булевых функций, налагая два ограничения на структуру двоичных разрешающих диаграмм.

R. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8):677–691, 1986.

1) в каждом пути из корня в терминальную вершину переменные должны следовать в одном и том же порядке.

2) в диаграмме не должно быть изоморфных поддеревьев или избыточных вершин.

Первое требование можно выполнить, задав линейный порядок $<$ на множестве переменных и объявив, что если вершина u имеет нетерминальную вершину–последователя v, то $var(u) < var(v)$.

Построение разрешающих диаграмм

Бриан показал, как построить канонические представления булевых функций, налагая два ограничения на структуру двоичных разрешающих диаграмм.

R. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8):677–691, 1986.

1) в каждом пути из корня в терминальную вершину переменные должны следовать в одном и том же порядке.

2) в диаграмме не должно быть изоморфных поддеревьев или избыточных вершин.

Первое требование можно выполнить, задав линейный порядок $<$ на множестве переменных и объявив, что если вершина u имеет нетерминальную вершину–последователя v, то $var(u) < var(v)$.

Второе требование можно выполнить, последовательно применяя следующие три правила преобразования BDDs.

Построение разрешающих диаграмм

Удаление повторных вхождений терминалов. Удаляются все кроме одной терминальные вершины, имеющие заданную пометку; все дуги, которые ранее вели в изъятые терминальные вершины, направляются теперь в оставшуюся терминальную вершину.

Построение разрешающих диаграмм

Удаление повторных вхождений терминалов. Удаляются все кроме одной терминальные вершины, имеющие заданную пометку; все дуги, которые ранее вели в изъятые терминальные вершины, направляются теперь в оставшуюся терминальную вершину.

Удаление повторных вхождений нетерминалов. Если для двух нетерминальных вершин u и v верны равенства $var(u) = var(v)$, $low(u) = low(v)$ и $high(u) = high(v)$, то одна из них удаляется, а все ведущие в нее дуги направляются в оставшуюся вершину.

Построение разрешающих диаграмм

Удаление повторных вхождений терминалов. Удаляются все кроме одной терминальные вершины, имеющие заданную пометку; все дуги, которые ранее вели в изъятые терминальные вершины, направляются теперь в оставшуюся терминальную вершину.

Удаление повторных вхождений нетерминалов. Если для двух нетерминальных вершин u и v верны равенства $var(u) = var(v)$, $low(u) = low(v)$ и $high(u) = high(v)$, то одна из них удаляется, а все ведущие в нее дуги направляются в оставшуюся вершину.

Удаление избыточных проверок Если для нетерминальной вершины v верно равенство $low(v) = high(v)$, то она удаляется, а все ведущие в нее дуги направляются в $low(v)$.

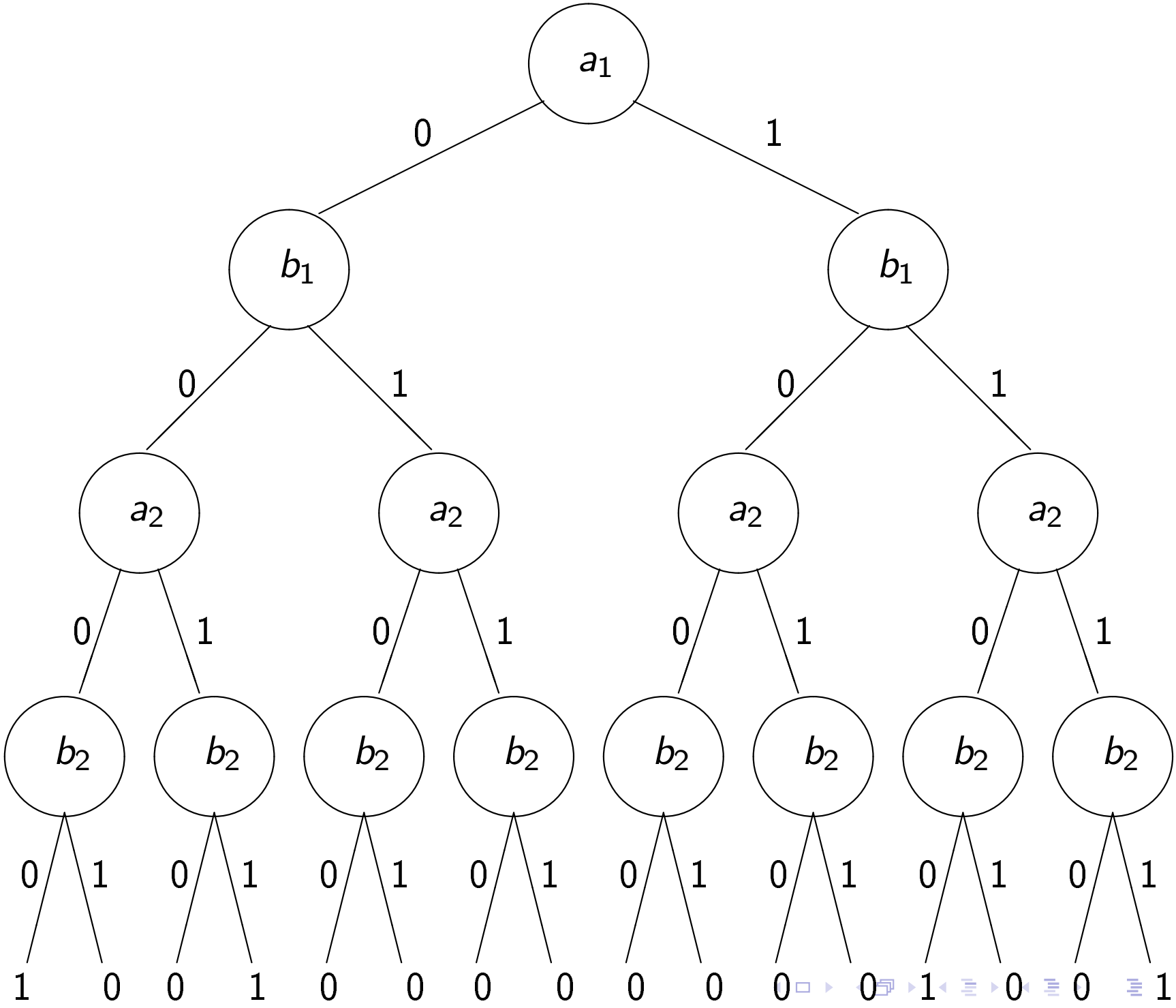
Построение разрешающих диаграмм

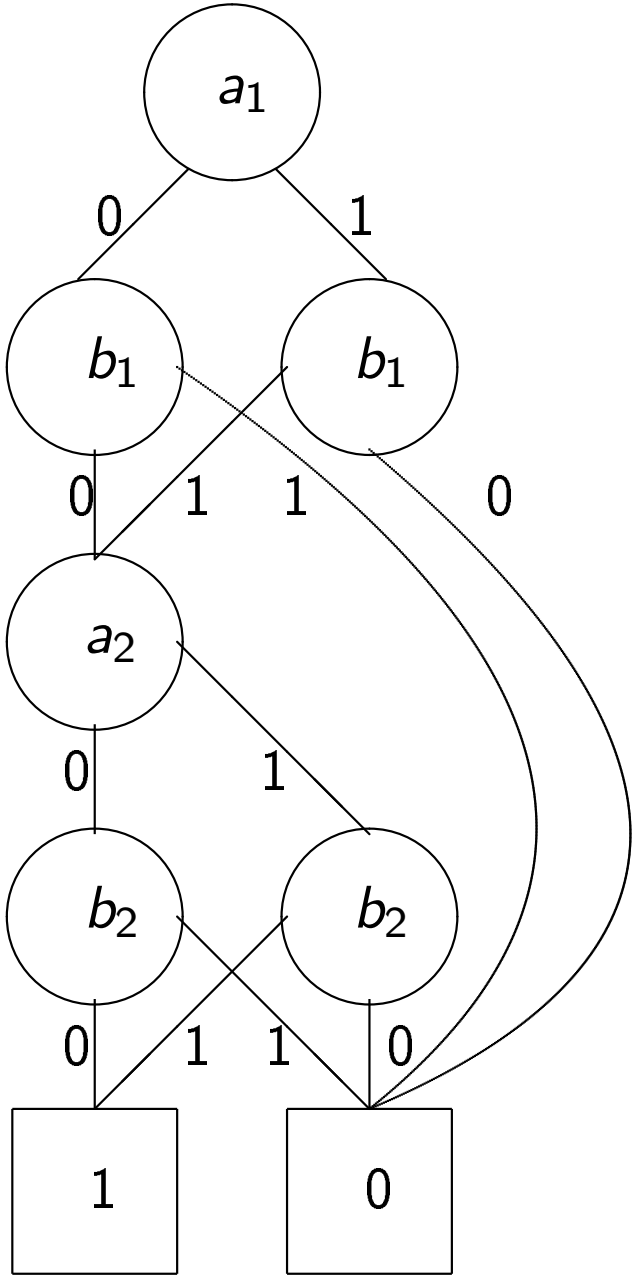
Каноническую форму можно получить из любой BDD, обладающей свойством упорядоченности; в ходе построения правила преобразования применяются до тех пор, пока они применимы.

BDD называется редуцированной упорядоченной двоичной разрешающей диаграммой (ROBDD), если ни одно из указанных правил к ней не применимо.

Бриан показал, как построить ROBDD, проводя преобразования произвольной BDD снизу вверх при помощи процедуры *Reduce* за время пропорциональное размеру исходной двоичной разрешающей диаграммы.

Например, если используется упорядочение $a_1 < b_1 < a_2 < b_2$ для двухбитовой функции сравнения, то в результате будет получена вот такая OBDD.


a_1
0
1
b_1
b_1
0
1
0
1
a_2
a_2
a_2
a_2
0
1
0
1
0
1
0
1
b_2
b_2
b_2
b_2
b_2
b_2
b_2
b_2
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
1
0
0
1
0
0
0
0
0
0
0
0
1
0
0
1


a_1
0
1
b_1
b_1
0
1
1
0
a_2
0
1
b_2
b_2
0
1
1
0
1
0

Построение разрешающих диаграмм

Если использовать ROBDD в качестве канонической формы для представления булевых функций, то проверка эквивалентности сводится к проверке изоморфизма между двоичными разрешающими диаграммами.

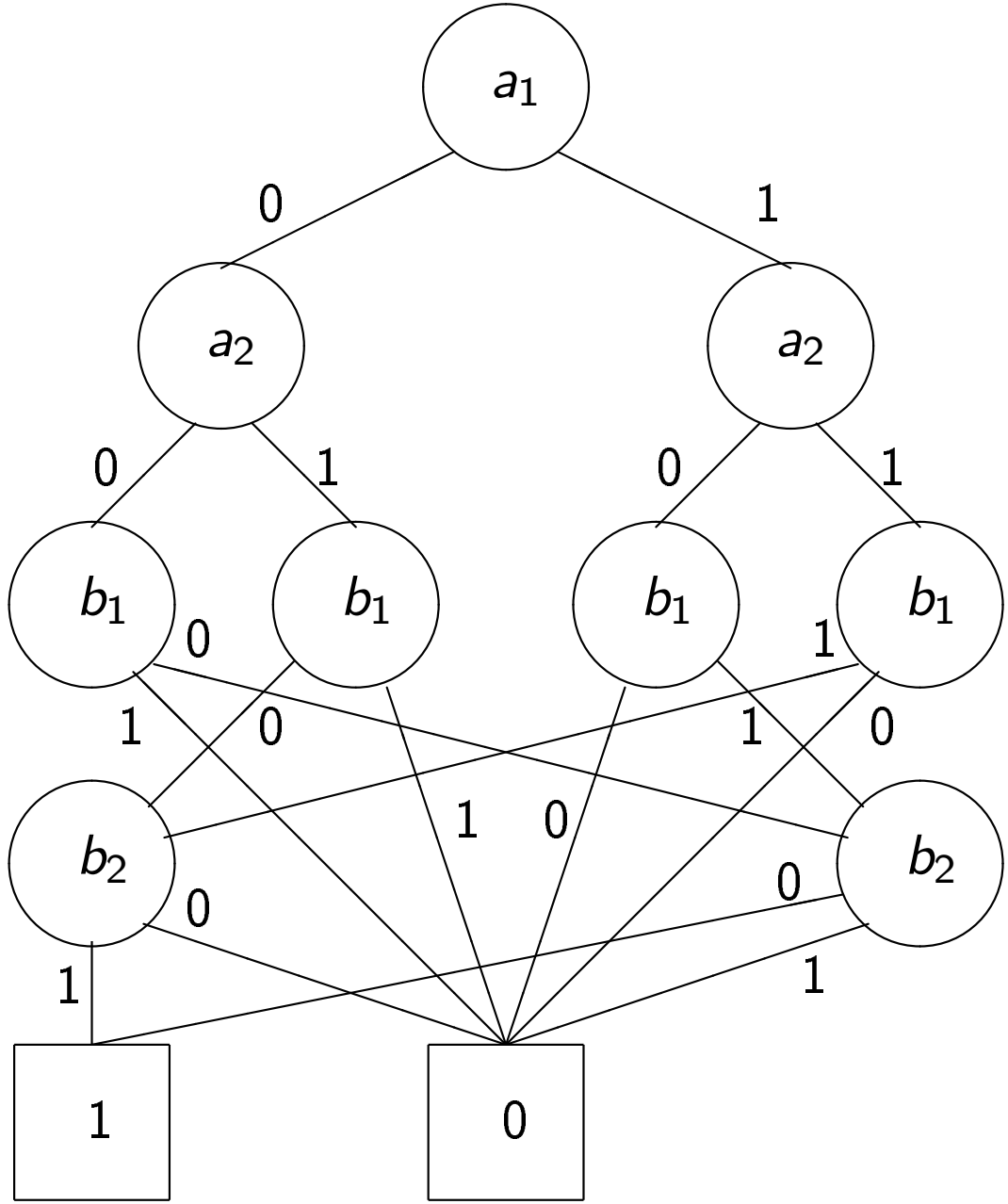
Это означает, что выполнимость может быть установлена путем проверки эквивалентности исследуемой ROBDD и вырожденной ROBDD, состоящей из единственной терминальной вершины, помеченной 0 .

Построение разрешающих диаграмм

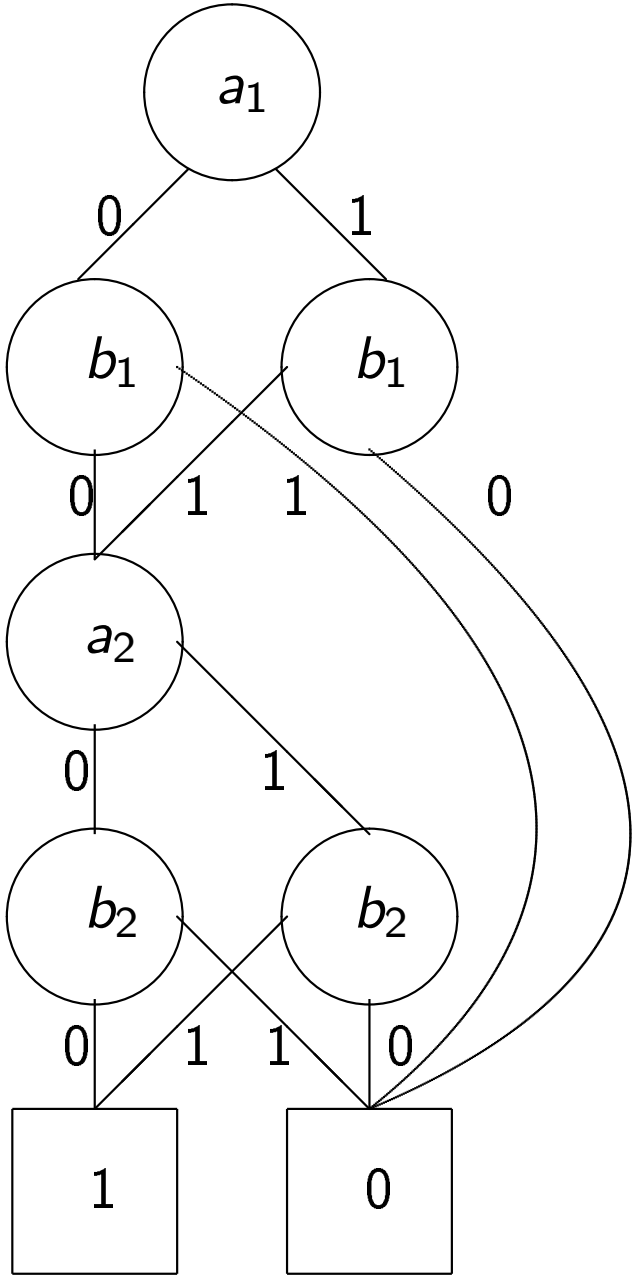
Если использовать ROBDD в качестве канонической формы для представления булевых функций, то проверка эквивалентности сводится к проверке изоморфизма между двоичными разрешающими диаграммами.

Это означает, что выполнимость может быть установлена путем проверки эквивалентности исследуемой ROBDD и вырожденной ROBDD, состоящей из единственной терминальной вершины, помеченной 0 .

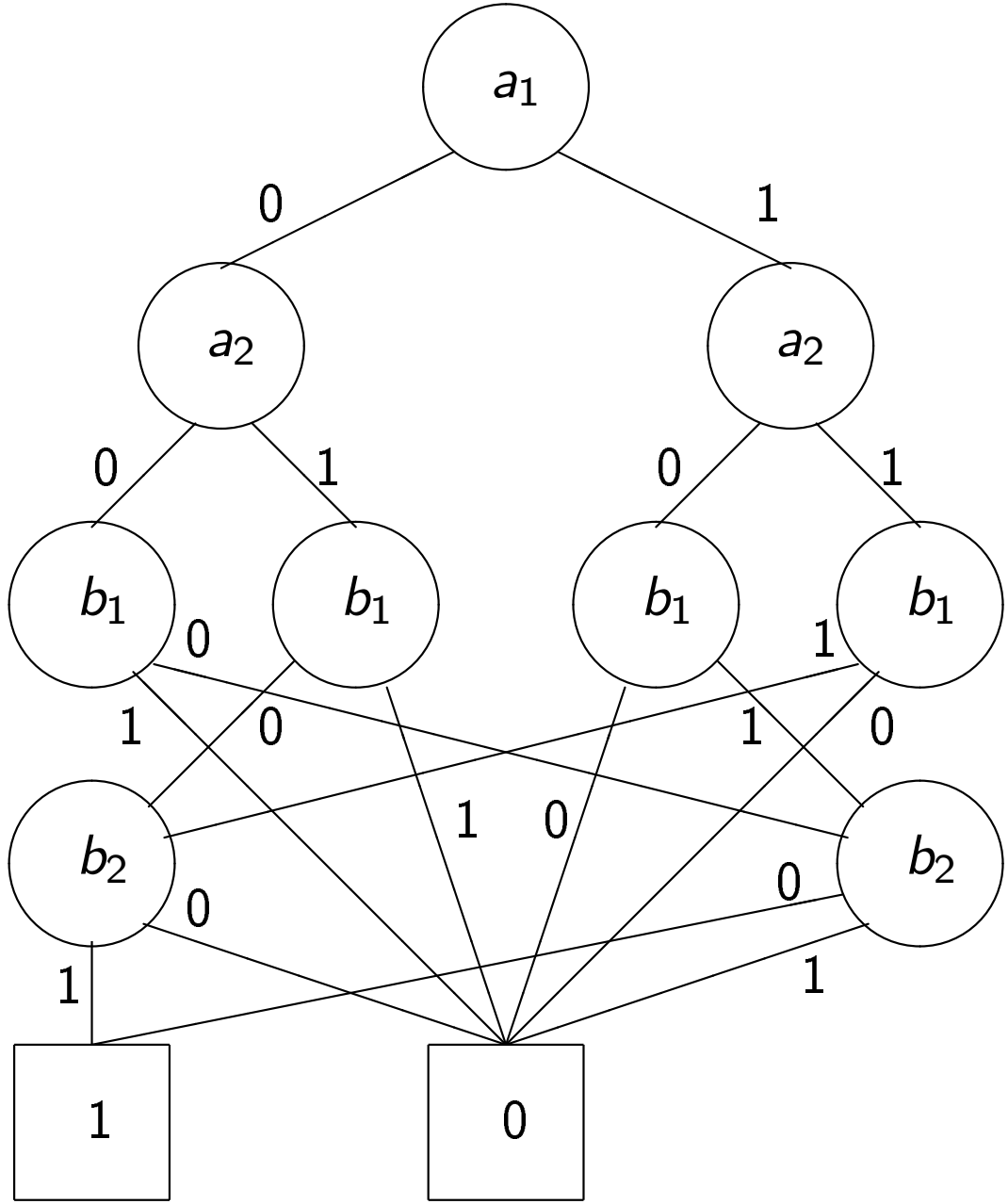
Размер OBDD очень сильно зависит от упорядочения переменных.
Например, если мы упорядочим переменные $a_1 < a_2 < b_1 < b_2$ для двухбитовой функции сравнения, то в результате будет получена вот такая ROBDD.



б) $a_1 < a_2 < b_1 < b_2$



а) $a_1 < b_1 < a_2 < b_2$



б) $a_1 < a_2 < b_1 < b_2$

Построение разрешающих диаграмм

В общем случае, когда речь идет о n-битовом компараторе, выбрав упорядочение переменных $a_1 < b_1 < \cdots < a_n < b_n$, мы получим ROBDD, имеющую $3n + 2$ вершин.

Но если выбрать упорядочение $a_1 < \cdots < a_n < b_1 \cdots < b_n$, то ROBDD будет иметь $3 \cdot 2^n - 1$ вершин.

Вообще говоря, проверка того, будет ли предложенный порядок оптимальным, является NP-полной задачей.

Более того, существуют последовательности булевых функций, размер OBDD для которых растет экспоненциально с увеличением числа переменных независимо от порядка их расположения.

Примером может служить функция, вычисляющая n-й бит произведения двоичных целых чисел.

Построение разрешающих диаграмм

Далее выясним, как выполнять различные логические операции для булевых функций, представленных в виде OBDD.

Построение разрешающих диаграмм

Далее выясним, как выполнять различные логические операции для булевых функций, представленных в виде OBDD.

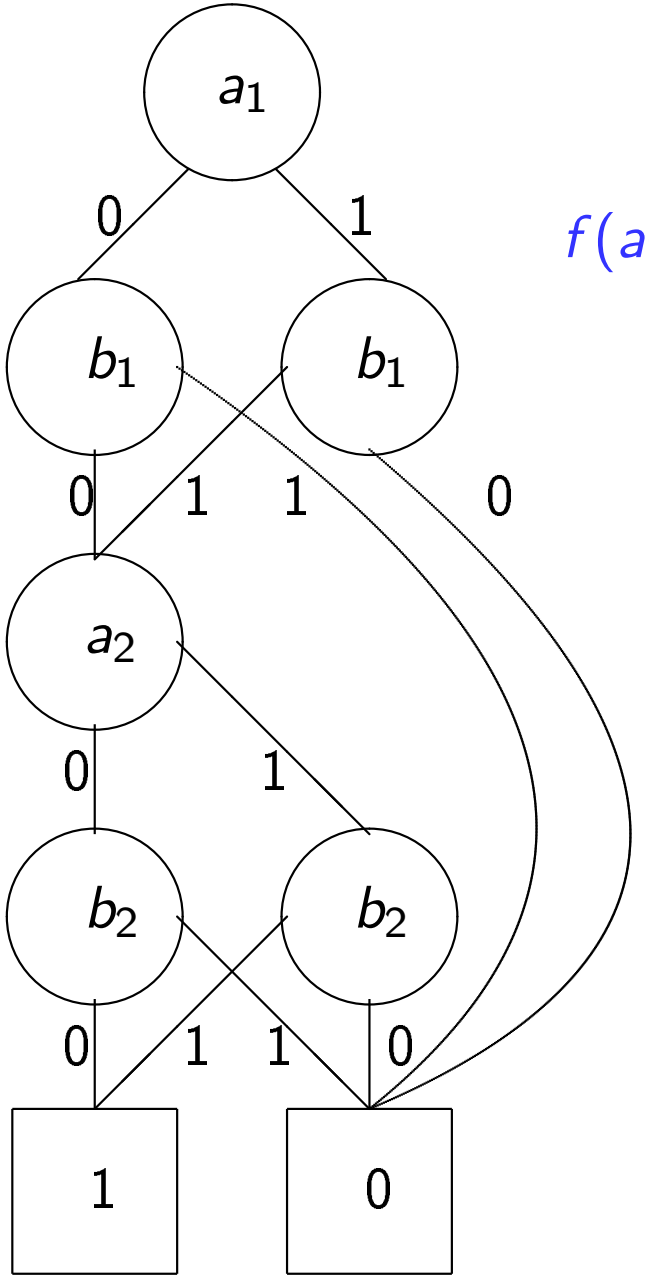
Начнем с операции подстановки константы b вместо заданной переменной x_i в заданной булевой функции f. Полученная в результате функция обозначается $f|_{x_i \leftarrow b}$ и определяется следующим соотношением:

$f(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$.

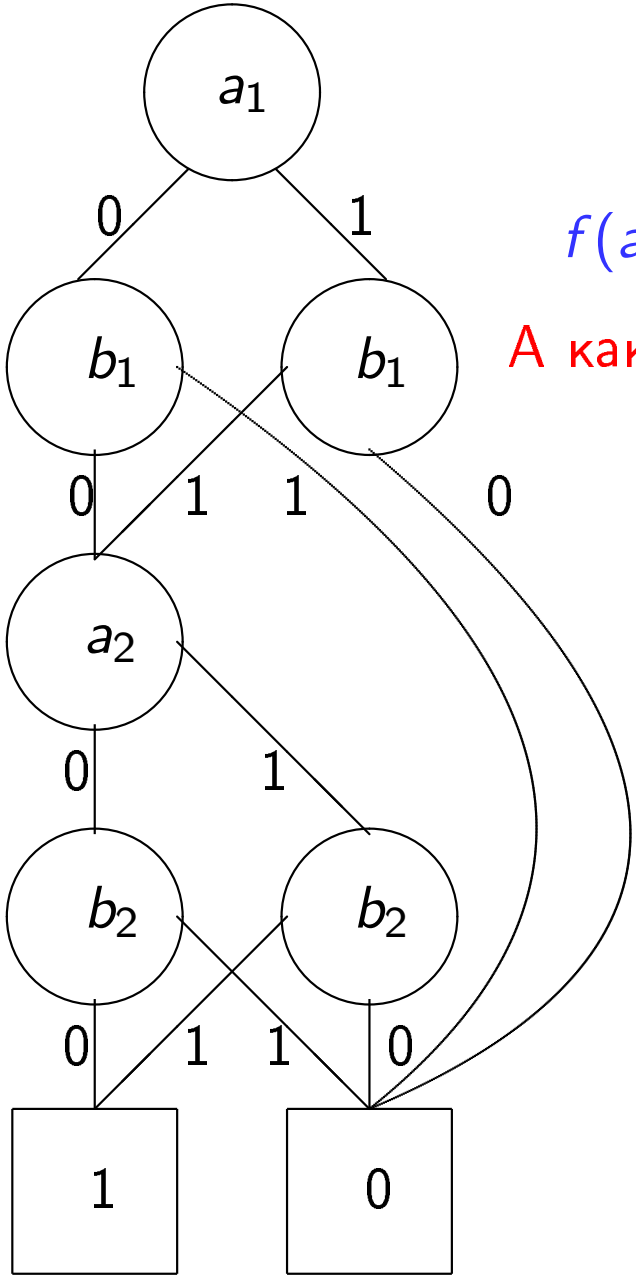
Если имеется ROBDD для f, то ROBDD для $f|_{x_i \leftarrow b}$ можно построить путем обхода исходной OBDD.

Если из вершины v дуга ведет в такую вершину w, что $var(w) = x_i$, то перенаправим эту дугу в вершину $low(w)$ в случае $b = 0$, и в $high(w)$ в случае $b = 1$.

Т.к. полученная OBDD может оказаться неканонической, применим процедуру *Reduce* для получения ROBDD, представляющую $f|_{x_i \leftarrow b}$.

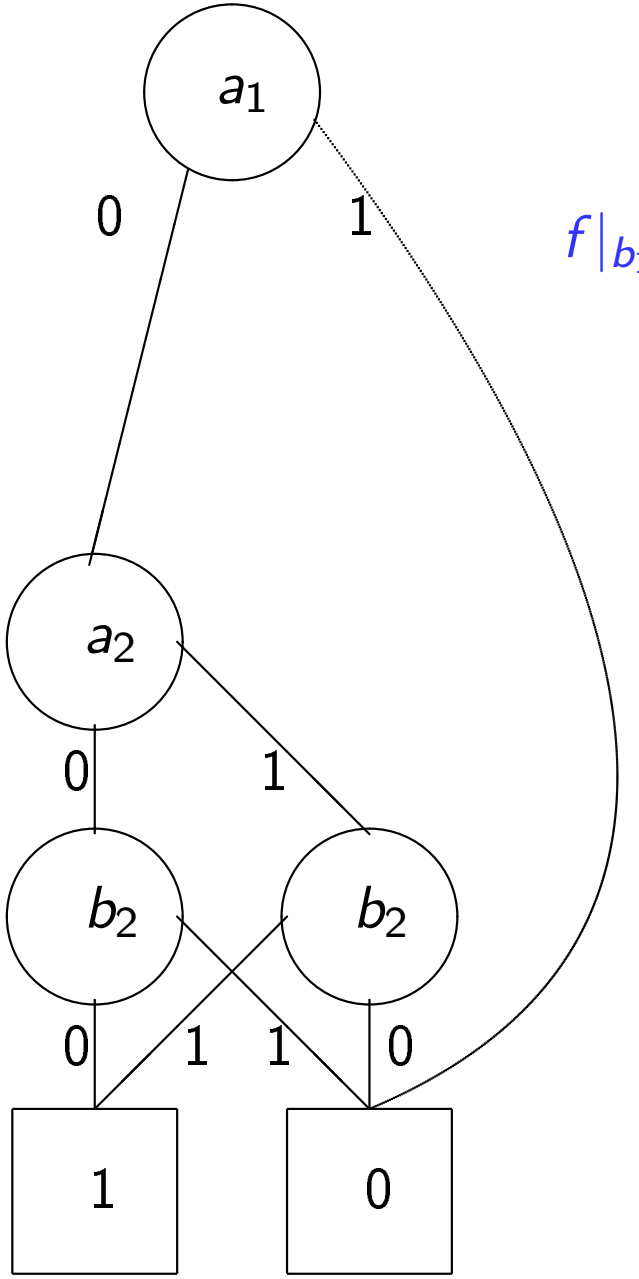


$f(a_1, a_2, b_1, b_2)$



$f(a_1, a_2, b_1, b_2)$

А как выглядит ROBDD для $f|_{b_1 \leftarrow 0}(a_1, a_2, b_1, b_2)$?



$f|_{b_1 \leftarrow 0}(a_1, a_2, b_1, b_2)$

Построение разрешающих диаграмм

Для булевых функций, представленных ROBDD, все шестнадцать двуместных булевых операций могут быть эффективно реализованы.
Сложность вычисления этих операций линейна относительно произведения размеров исходных OBDD.

Ключом к их реализации служит разложение Шеннона

$$f = \left(\neg x \wedge f\big|_{x \leftarrow 0}\right) \vee \left(x \wedge f\big|_{x \leftarrow 1}\right)$$

Бриан предложил единый алгоритм *Apply* для вычисления всех шестнадцати операций.
Алгоритм строится методом динамического программирования.
Поясним вкратце, как он работает.

Построение разрешающих диаграмм

Обозначим символом $*$ произвольную двухместную логическую операцию и рассмотрим пару булевых функций f и f'.

Введем следующие обозначения:

- v и v' — корневые вершины OBDD для f и f';
- $x = var(v)$ и $x' = var(v')$.

Построение разрешающих диаграмм

Возможны несколько вариантов в зависимости от соотношений между v и v'.

Построение разрешающих диаграмм

Возможны несколько вариантов в зависимости от соотношений между v и v'.

- Если обе вершины v и v' терминальные, то $f * f' = value(v) * value(v')$.

Построение разрешающих диаграмм

Возможны несколько вариантов в зависимости от соотношений между v и v'.

- Если обе вершины v и v' терминальные, то $f * f' = value(v) * value(v')$.
- Если $x = x'$, то мы воспользуемся разложением Шеннона

$$f * f' = (\neg x \wedge (f|_{x \leftarrow 0} * f'|_{x \leftarrow 0})) \vee (x \wedge (f|_{x \leftarrow 1} * f'|_{x \leftarrow 1}))$$

для разбиения задачи на две подзадачи, которые решаются рекурсивно.
Корневой вершиной результирующей ROBDD будет новая вершина w, для которой $var(w) = x$;
при этом $low(w)$ — это корневая вершина ROBDD для $f|_{x \leftarrow 0} * f'|_{x \leftarrow 0}$, а $high(w)$ — это корневая вершиной ROBDD для $f|_{x \leftarrow 1} * f'|_{x \leftarrow 1}$.

Построение разрешающих диаграмм

- Если $x < x'$, то $f'|_{x \leftarrow 0} = f'|_{x \leftarrow 1} = f'$, поскольку f' не зависит от x. В таком случае разложение Шеннона упрощается и принимает вид

$$f * f' = (\neg x \wedge (f|_{x \leftarrow 0} * f')) \vee (x \wedge (f|_{x \leftarrow 1} * f')),$$

а ROBDD для $f * f'$ вычисляется рекурсивно, как и в предыдущем случае.

Построение разрешающих диаграмм

- Если $x < x'$, то $f'|_{x \leftarrow 0} = f'|_{x \leftarrow 1} = f'$, поскольку f' не зависит от x. В таком случае разложение Шеннона упрощается и принимает вид

$$f * f' = (\neg x \wedge (f|_{x \leftarrow 0} * f')) \vee (x \wedge (f|_{x \leftarrow 1} * f')),$$

  а ROBDD для $f * f'$ вычисляется рекурсивно, как и в предыдущем случае.
- Если $x' < x$, то схема вычислений будет аналогична предыдущему случаю.

Построение разрешающих диаграмм

Чтобы предотвратить экспоненциальное разрастание вычислений, применяется метод динамического программирования.

Каждая подзадача соответствует паре ROBDD, являющихся подграфами исходных ROBDD для f и f'. Таким образом, количество подзадач ограничено произведением размеров OBDD для f и f'.

Хэш-таблица под названием кэш результатов используется для хранения в памяти ранее вычисленных подзадач.
Перед рекурсивным обращением, мы просматриваем этот кэш и проверяем, была ли уже решена предъявленная подзадача. Если это так, то результат выбирается из кэша; иначе выполняется рекурсивное обращение, и построенная ROBDD помещается в кэш результатов.

Построение разрешающих диаграмм

Булево отрицание можно реализовать, применяя *Apply*. Но проще вычислить ROBDD для $\neg f$, поменяв местами значения в терминальных вершинах OBDD для f .

Построение разрешающих диаграмм

Булево отрицание можно реализовать, применяя *Apply*. Но проще вычислить ROBDD для $\neg f$, поменяв местами значения в терминальных вершинах OBDD для f .

Был разработан ряд обобщений этого метода для представления семейства булевых функций, имеющих общие подграфы.
Для всех функций этого семейства действует один и тот же порядок расположения переменных.
Если используется подобное обобщение, то две функции семейства будут равны в том и только том случае, когда они имеют один и тот же корень.
Следовательно, время, требуемое для проверки равенства двух функций, ограничено постоянной величиной.

Современные пакеты OBDD могут эффективно работать с графами, имеющими миллионы вершин.

ROBDD представление моделей Крипке

OBDD удобны для получения компактных представлений отношений на конечных областях.

Всякое n-местное отношение Q на множестве $\{0, 1\}$ можно представить в виде OBDD для его характеристической функции:

$$f_Q(x_1, \ldots, x_n) = 1 \iff Q(x_1, \ldots, x_n) = \mathit{true}.$$

ROBDD представление моделей Крипке

Чтобы построить ROBDD представление n-местного отношения Q на конечной области D, содержащей не более 2^m элементов, нужно закодировать все элементы множества D двоичными векторами, воспользовавшись какой-нибудь биекцией $\phi : \{0,1\}^m \to D$.

Далее определяется булево отношение $\widehat{Q}$ местности $m \times n$ по следующему правилу:

$$\widehat{Q}(\overline{x}_1, \ldots, \overline{x}_n) = Q(\phi(\overline{x}_1), \ldots, \phi(\overline{x}_n)),$$

где $\overline{x}_i$ — вектор, состоящий из m булевых переменных, при помощи которого закодирована переменная x_i, принимающая значения из D.

Теперь отношение Q можно представить в виде ROBDD для характеристической функции $f_{\widehat{Q}}$ отношения $\widehat{Q}$.

ROBDD представление моделей Крипке

Чтобы отыскать ROBDD представление модели Крипке $M = (S, R, L)$, нужно описать множество S, отношение R и отображение L.

Закодируем состояния множества S булевыми векторами длины m.
Используем функцию $\phi : \{0,1\}^m \rightarrow S$ для отображения булевых векторов в множество состояний S.

ROBDD представление моделей Крипке

Чтобы отыскать ROBDD представление модели Крипке $M = (S, R, L)$, нужно описать множество S, отношение R и отображение L.

Закодируем состояния множества S булевыми векторами длины m.
Используем функцию $\phi : \{0,1\}^m \rightarrow S$ для отображения булевых векторов в множество состояний S.

Для представления отношения переходов потребуются два комплекта булевых переменных: одно для представления состояния, которым начинается переход, а другое для представления состояния, в котором переход оканчивается.

Если отношение переходов R закодировано булевой функцией $\widehat{R}(\overline{x}, \overline{x}')$, то отношение R задается OBDD для характеристической функции $f_{\widehat{R}}$.

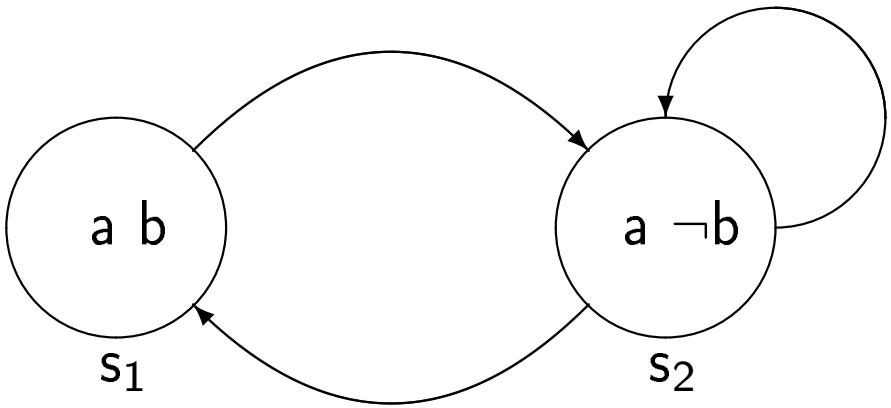
Представление моделей Крипке

Функцию разметки L можно рассматривать как отображение из множества атомарных высказываний в множество всех подмножеств пространства состояний S : каждое атомарное высказывание p отображается в множество $L_p = \{s \mid p \in L(s)\}$ тех состояний, в которых оно выполняется.

Множества L_p можно представить в виде OBDD при помощи кодирования ϕ .

Представление моделей Крипке

Рассмотрим модель с двумя состояниями



Переход из состояния s_1 в состояние s_2 будем представлять конъюнкцией

$$(a \wedge b \wedge a' \wedge \neg b').$$

Булева формула для всего отношения переходов имеет вид

$$(a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge b').$$

Эта формула содержит три дизъюнктивных слагаемых, поскольку модель Крипке содержит три перехода. Далее эта формула преобразуется в OBDD для наиболее компактного представления отношения переходов.

Упражнения

Рассмотрите все возможные упорядочения переменных x, y, z и для каждого из них постройте упорядоченные двоичные разрешающие диаграммы, являющиеся каноническими представлениями функций $f(x, y, z)$ и $g(x, y, z)$, которые заданы следующими таблицами.

x	y	z	$f(x, y, z)$	$g(x, y, z)$
1	1	1	0	1
1	1	0	1	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0

Упражнения

Выберите наилучшее упорядочение переменных x, y, z и постройте OBDD, являющиеся представлением следующих функций

- $F_1(x, y, z) = \neg f(x, y, z)$;
- $F_2(x, y, z) = f(x, y, z) \wedge g(x, y, z)$;
- $F_3(x, y, z) = f(x, y, z) \vee g(x, y, z)$;
- $F_4(x, y, z) = F_2(x, y, z) \oplus F_3(x, y, z)$.

Проверьте, является ли выбранный Вами порядок расположения переменных оптимальным для представления перечисленных функций в виде OBDD.

Упражнения

Напишите программы, реализующие алгоритмы *Reduce* и *Apply*.

КОНЕЦ ЛЕКЦИИ 5.