

Оглавление

[Введение 3](#_Toc374278002)

[Постановка задачи 3](#_Toc374278003)

[Реализация 4](#_Toc374278004)

[Основная идея 4](#_Toc374278005)

[Модель на языке Promela 4](#_Toc374278006)

[LTL правила 7](#_Toc374278007)

[Безопасность 7](#_Toc374278008)

[Живость и справедливость 8](#_Toc374278009)

[Результаты моделирования 9](#_Toc374278010)

[Выводы 10](#_Toc374278011)

[Список литературы 10](#_Toc374278012)

# Введение

От программных систем все в большей степени каждодневно зависят жизнь и здоровье людей, однако программирование до сих пор остается единственной областью инженерной деятельности, где разработчик фактически не может гарантировать качество своей работы[1].

Верификация – это набор формальных приемов и методов подтверждения того, что разрабатываемая система удовлетворяет формальным установленным требованиям (формальной спецификации)[2].

В данной курсовой работе рассматривается модель контроллера управления движением на дорожном перекрестке. Необходимо описать модель и проверить ее корректность при выполнении следующих условий:

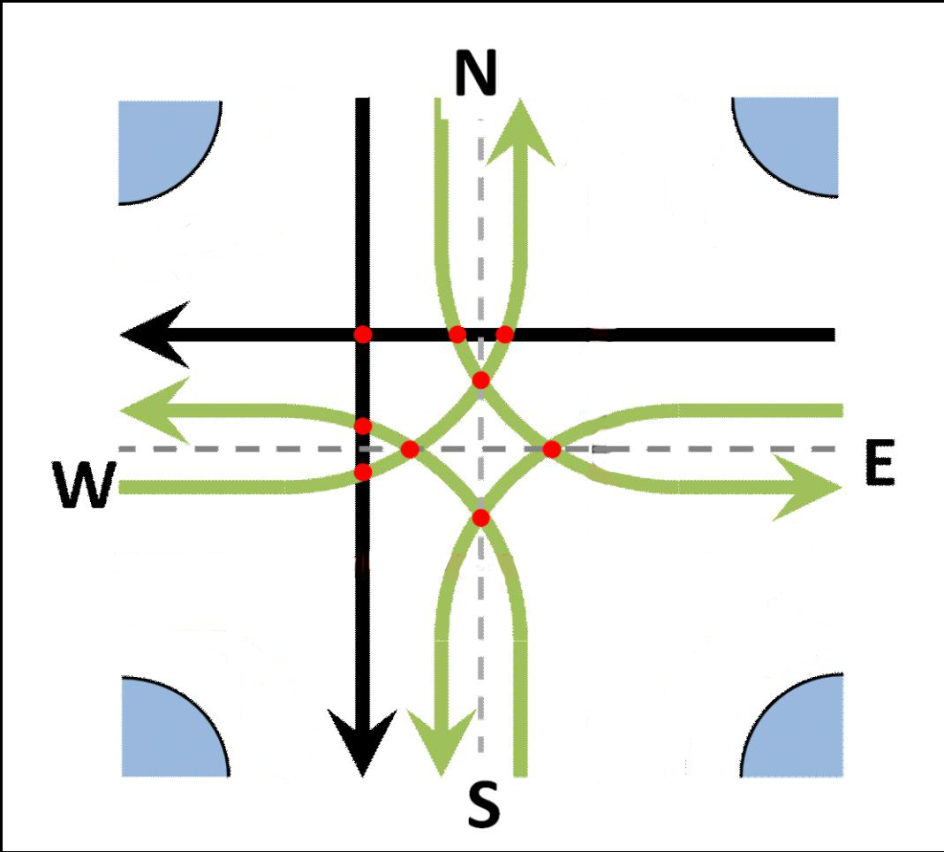
* Каждое направление контролирует отдельный светофор
* Поведение светофоров описывается параллельными процессами
* Необходимо моделировать появление машин
* Алгоритм управления движением не должен определяться заранее заданным порядком переключения светофоров
* В системе для каждого из направлений присутствуют датчики, фиксирующие наличие автомобилей.

При этом для model checking используется система Spin. Данная система позволяет описывать модель на языке Promela и задавать требуемые свойства системы при помощи выражений LTL. Такой подход позволяет проверить корректность модели перед ее реализацией.

# Постановка задачи

Вариант (1, 12, 15).

Пересечения: {(NS, WN), (NE, EW),(SW,ES)}



Для заданного варианта необходимо создать набор свойств (LTL), которые определяют корректность работы модели (например отсутствие варианта, когда движение разрешено всем), а так же описать саму модель на языке Promela.

# Реализация

## Основная идея

Основной идеей данной реализации является отождествление пересечений направлений движения с ресурсами, которые необходимо получить контролеру для разрешения безопасного проезда. Данных подход обладает целым рядом преимуществ, из которых можно выделить:

1. Контролеры, управляющие пересекающимися направлениями, не могут дать зеленый свет одновременно
2. «Дружественные» контроллеры имеют возможность пропускать поток независимо
3. Данный подход довольно просто в реализации

## Модель на языке Promela

Данная модель была построена на основе следующих постулатов:

1. Каждый контроллер светофора является отдельным процессом.
2. Сигнал светофора может быть двух видов: красный – зеленый
3. Состояние светофора описывается глобальными переменными
4. Датчики движения также являются глобальными переменными
5. Движение (траффик машин) генерируется внешним, по отношению к контроллерам, процессом случайным образом

Реализация:

/\* Course work made by Anton Lukashin group 6084-12 \*/

/\* Exercise 1,12,15 (WN,NS) (NE, EW) (SW, ES)\*/

/\* Types of signals \*/

mtype = {Red, Green};

/\* Lights signals \*/

mtype NS\_L = Red;

mtype WN\_L = Red;

mtype NE\_L = Red;

mtype EW\_L = Red;

mtype ES\_L = Red;

mtype SW\_L = Red;

/\* Car traffic sensors \*/

bool NS\_S = false;

bool WN\_S = false;

bool NE\_S = false;

bool EW\_S = false;

bool ES\_S = false;

bool SW\_S = false;

#define pNS\_S (NS\_L==Green && WN\_L==Green && SW\_L==Green && EW\_L==Green)

#define pWN\_S (WN\_L==Green && NS\_L==Green && NE\_L==Green && SW\_L==Green && EW\_L==Green)

#define pEW\_S (EW\_L==Green && NE\_L==Green && WN\_L==Green && NS\_L==Green)

#define pNE\_S (NE\_L==Green && WN\_L==Green && ES\_L==Green && EW\_L==Green)

#define pSW\_S (SW\_L==Green && ES\_L==Green && WN\_L==Green && NS\_L==Green)

#define pES\_S (SW\_L==Green && ES\_L==Green && WN\_L==Green && NS\_L==Green)

#define pNS\_L (NS\_S && (NS\_L==Red))

#define qNS\_L (NS\_L==Green)

#define pWN\_L (WN\_S && (WN\_L==Red))

#define qWN\_L (WN\_L==Green)

#define pEW\_L (EW\_S && (EW\_L==Red))

#define qEW\_L (EW\_L==Green)

#define pNE\_L (NE\_S && (NE\_L==Red))

#define qNE\_L (NE\_L==Green)

#define pSW\_L (SW\_S && (SW\_L==Red))

#define qSW\_L (SW\_L==Green)

#define pES\_L (ES\_S && (ES\_L==Red))

#define qES\_L (ES\_L==Green)

#define pNS\_F ((NS\_L==Green) && NS\_S)

#define pWN\_F ((WN\_L==Green) && WN\_S)

#define pEW\_F ((EW\_L==Green) && EW\_S)

#define pNE\_F ((NE\_L==Green) && NE\_S)

#define pSW\_F ((SW\_L==Green) && SW\_S)

#define pES\_F ((ES\_L==Green) && ES\_S)

/\*Safety\*/

ltl p0\_1 {[] !pNS\_S}

ltl p0\_2 {[] !pWN\_S}

ltl p0\_3 {[] !pEW\_S}

ltl p0\_4 {[] !pNE\_S}

ltl p0\_5 {[] !pSW\_S}

ltl p0\_6 {[] !pES\_S}

/\*Liveness\*/

ltl p1\_1 {[] (!pNS\_L || (<> qNS\_L))}

ltl p1\_2 {[] (pWN\_L -> (<> qWN\_L))}

ltl p1\_3 {[] (pEW\_L -> (<> qEW\_L))}

ltl p1\_4 {[] (pNE\_L -> (<> qNE\_L))}

ltl p1\_5 {[] (pSW\_L -> (<> qSW\_L))}

ltl p1\_6 {[] (pES\_L -> (<> qES\_L))}

/\*Fairness\*/

ltl p2\_1 {[] <> !pNS\_F -> ([] <> (!pNS\_F) && (pNS\_L -> (<> qNS\_L)))}

ltl p2\_2 {[] <> !pWN\_F -> ([] <> (!pWN\_F) && (pWN\_L -> (<> qWN\_L)))}

ltl p2\_3 {[] <> !pEW\_F -> ([] <> (!pEW\_F) && (pEW\_L -> (<> qEW\_L)))}

ltl p2\_4 {[] <> !pNE\_F -> ([] <> (!pNE\_F) && (pNE\_L -> (<> qNE\_L)))}

ltl p2\_5 {[] <> !pES\_F -> ([] <> (!pES\_F) && (pES\_L -> (<> qES\_L)))}

ltl p2\_6 {[] <> !pSW\_F -> ([] <> (!pSW\_F) && (pSW\_L -> (<> qSW\_L)))}

/\* Synchronization channels \*/

chan NS\_WN\_EW = [1] of {bool};

chan NS\_WN\_SW = [1] of {bool};

chan NE\_WN\_EW = [1] of {bool};

chan NE\_ES = [1] of {bool};

chan ES\_SW = [1] of {bool};

init

{

atomic{

NS\_WN\_EW ! true;

NS\_WN\_SW ! true;

NE\_WN\_EW ! true;

NE\_ES ! true;

ES\_SW ! true;

};

atomic{

run NS();

run WN();

run NE();

run ES();

run EW();

run SW();

run gen\_t();

};

}

/\* Traffic generation process \*/

proctype gen\_t ()

{

accpet: do

:: true->

if

:: (NS\_L==Green || !NS\_S) -> NS\_S = !NS\_S;

:: else -> skip;

fi

:: true->

if

:: (WN\_L==Green || !WN\_S) -> WN\_S = !WN\_S;

:: else -> skip;

fi

:: true->

if

:: (NE\_L==Green || !NE\_S) -> NE\_S = !NE\_S;

:: else -> skip;

fi

:: true->

if

:: (EW\_L==Green || !EW\_S) -> EW\_S = !EW\_S;

:: else -> skip;

fi

:: true->

if

:: (ES\_L==Green || !ES\_S) -> ES\_S = !ES\_S;

:: else -> skip;

fi

:: true->

if

:: (SW\_L==Green || !SW\_S) -> SW\_S = !SW\_S;

:: else -> skip;

fi

od;

}

/\* NS controller \*/

proctype NS ()

{

end:

do

:: NS\_S ->

/\* Wait for resources \*/

NS\_WN\_EW ? true; NS\_WN\_SW ? true;

NS\_L = Green;

if

/\* Wait for end of car queue \*/

:: !NS\_S -> skip;

fi;

NS\_L = Red;

NS\_WN\_EW ! true; NS\_WN\_SW ! true;

od;

}

/\* WN controller \*/

proctype WN ()

{

end:

do

:: WN\_S ->

/\* Wait for resources \*/

NS\_WN\_EW ? true; NS\_WN\_SW ? true; NE\_WN\_EW ? true;

WN\_L = Green;

if

/\* Wait for end of car queue \*/

:: !WN\_S -> skip;

fi;

WN\_L = Red;

NS\_WN\_EW ! true; NS\_WN\_SW ! true; NE\_WN\_EW ! true;

od;

}

/\* NE controller \*/

proctype NE ()

{

end:

do

:: NE\_S ->

/\* Wait for resources \*/

NE\_WN\_EW ? true; NE\_ES ? true;

NE\_L = Green;

if

/\* Wait for end of car queue \*/

:: !NE\_S -> skip;

fi;

NE\_L = Red;

NE\_WN\_EW ! true; NE\_ES ! true;

od;

}

/\* NE controller \*/

proctype EW ()

{

end:

do

:: EW\_S ->

/\* Wait for resources \*/

NS\_WN\_EW ? true; NE\_WN\_EW ? true;

EW\_L = Green;

if

/\* Wait for end of car queue \*/

:: !EW\_S -> skip;

fi;

EW\_L = Red;

NS\_WN\_EW ! true; NE\_WN\_EW ! true;

od;

}

/\* ES controller \*/

proctype ES ()

{

end:

do

:: ES\_S ->

/\* Wait for resources \*/

ES\_SW ? true; NE\_ES ? true;

ES\_L = Green;

if

/\* Wait for end of car queue \*/

:: !ES\_S -> skip;

fi;

ES\_L = Red;

ES\_SW ! true; NE\_ES ! true;

od;

}

/\* SW controller \*/

proctype SW ()

{

end:

do

:: SW\_S ->

/\* Wait for resources \*/

NS\_WN\_SW ? true; ES\_SW ? true;

SW\_L = Green;

if

/\* Wait for end of car queue \*/

:: !SW\_S -> skip;

fi;

SW\_L = Red;

NS\_WN\_SW ! true; ES\_SW ! true;

od;

}

## LTL правила

Данные правила на языке темпоральной логики оперируют двумя базовыми понятиями:

1. G ([] – в системе Spin) – описывает свойство, которое должно выполняться всегда
2. F (<> - в системе Spin) – описывает свойство, которое должно выполнится когда-то в будущем

Правила данной системы:

### Безопасность

Для данной модели правила безопасности звучат на естественном языке следующим образом: «Никогда не будет такой ситуации, что на данном направлении будет гореть зеленый свет, и на всех, пересекающих это направление дорогах, тоже будет зеленый»

1. **NS** - ltl p0\_1 {[] !pNS\_S}

pNS\_S (NS\_L==Green && WN\_L==Green && SW\_L==Green && EW\_L==Green)

1. **WN** – ltl p0\_2 {[] !pWN\_S}

pWN\_S (WN\_L==Green && NS\_L==Green && NE\_L==Green && SW\_L==Green && EW\_L==Green)

1. **NE** - ltl p0\_3 {[] !pEW\_S}

pEW\_S (EW\_L==Green && NE\_L==Green && WN\_L==Green && NS\_L==Green)

1. **EW** - ltl p0\_4 {[] !pNE\_S}

pNE\_S (NE\_L==Green && WN\_L==Green && ES\_L==Green && EW\_L==Green)

1. **SW** - ltl p0\_5 {[] !pSW\_S}

pSW\_S (SW\_L==Green && ES\_L==Green && WN\_L==Green && NS\_L==Green)

1. **ES** - ltl p0\_6 {[] !pES\_S}

pES\_S (SW\_L==Green && ES\_L==Green && WN\_L==Green && NS\_L==Green)

### Живость и справедливость

Для данной модели правила **живости (liveness)** звучат на естественном языке следующим образом: «Всегда выполняется, если светофор горит красным и датчик показывает наличие машин, следовательно в будущем данный светофор будет гореть зеленым». А правила **справедливости (fairness)** - «Невозможна такая ситуация, что в конкретном направлении движется непрерывный поток машин(т.е. светофор должен неопределенно часто менять свой сигнал на запрещающий)».

Доказывать свойства fairness и liveness отдельно для данной модели не представляется возможным из-за того, что система Spin обладает «случайным» планировщиком процессов, который не гарантирует выполнения всех процессов. Таким образом, может сложиться ситуация, при которой один или несколько процессов будут выполняться всегда, в то время как все остальные процессы будут простаивать. Ясно, что в такой ситуации свойства fairness и liveness для простаивающих процессов выполняться не будут. Таким образом, свойства fairness и liveness надо доказывать вместе. На естественном языке объединенное свойство будет звучать следующим образом: «При наличии ожидающих автомобилей на каком-либо направлении ему обязательно представится возможность проехать (возможно, через какое-то время), при ограничении, что в каждом направлении не движется непрерывный поток автомобилей».

1. **NS** - {[] <> !pNS\_F -> ([] <> (!pNS\_F) && (pNS\_L -> (<> qNS\_L)))}

pNS\_F ((NS\_L==Green) && NS\_S)

pNS\_L (NS\_S && (NS\_L==Red))

qNS\_L (NS\_L==Green)

1. **WN** - {[] <> !pWN\_F -> ([] <> (!pWN\_F) && (pWN\_L -> (<> qWN\_L)))}

pWN\_F ((WN\_L==Green) && WN\_S)

pWN\_L (WN\_S && (WN\_L==Red))

qWN\_L (WN\_L==Green)

1. **NE** - {[] <> !pEW\_F -> ([] <> (!pEW\_F) && (pEW\_L -> (<> qEW\_L)))}

pNE\_F ((NE\_L==Green) && NE\_S)

pNE\_L (NE\_S && (NE\_L==Red))

qNE\_L (NE\_L==Green)

1. **EW** - {[] <> !pNE\_F -> ([] <> (!pNE\_F) && (pNE\_L -> (<> qNE\_L)))}

pEW\_F ((EW\_L==Green) && EW\_S)

pEW\_L (EW\_S && (EW\_L==Red))

qEW\_L (EW\_L==Green)

1. **SW** - {[] <> !pES\_F -> ([] <> (!pES\_F) && (pES\_L -> (<> qES\_L)))}

pSW\_F ((SW\_L==Green) && SW\_S)

pSW\_L (SW\_S && (SW\_L==Red))

qSW\_L (SW\_L==Green)

1. **ES** - {[] <> !pSW\_F -> ([] <> (!pSW\_F) && (pSW\_L -> (<> qSW\_L)))}

pES\_F ((ES\_L==Green) && ES\_S)

pES\_L (ES\_S && (ES\_L==Red))

qES\_L (ES\_L==Green)

# Результаты моделирования

Моделирование показало, что все проверки на безопасность и живучесть при условии справедливости система проходит успешно. Результаты моделирования для одной из проверок(живучесть при условии справедливости для направления NS) приведены ниже:

gcc -DMEMLIM=4500 -O2 -DXUSAFE -w -o pan pan.c

./pan -m10000000 -a -N p2\_1

Pid: 2161

Depth= 406709 States= 1e+06 Transitions= 2.9e+06 Memory= 723.972 t=1.48 R=7e+05

Depth= 1000496 States= 2e+06 Transitions= 6.28e+06 Memory= 800.925 t=3.29 R=6e+05

Depth= 1533149 States= 3e+06 Transitions= 9.9e+06 Memory= 880.124 t=5.27 R=6e+05

Depth= 2070939 States= 4e+06 Transitions= 1.36e+07 Memory= 963.327 t=7.35 R=5e+05

Depth= 2528871 States= 5e+06 Transitions= 1.74e+07 Memory= 1046.628 t=9.53 R=5e+05

Depth= 2941845 States= 6e+06 Transitions= 2.12e+07 Memory= 1129.636 t=11.7 R=5e+05

Depth= 3341183 States= 7e+06 Transitions= 2.51e+07 Memory= 1215.671 t=14 R=5e+05

Depth= 3707041 States= 8e+06 Transitions= 2.89e+07 Memory= 1302.097 t=16.4 R=5e+05

Depth= 4068597 States= 9e+06 Transitions= 3.28e+07 Memory= 1389.499 t=18.8 R=5e+05

Depth= 4413393 States= 1e+07 Transitions= 3.66e+07 Memory= 1475.925 t=21.2 R=5e+05

Depth= 4776589 States= 1.1e+07 Transitions= 4.04e+07 Memory= 1562.741 t=23.6 R=5e+05

Depth= 5128173 States= 1.2e+07 Transitions= 4.41e+07 Memory= 1648.484 t=26.1 R=5e+05

Depth= 5442981 States= 1.3e+07 Transitions= 4.79e+07 Memory= 1734.812 t=28.5 R=5e+05

Depth= 5807473 States= 1.4e+07 Transitions= 5.17e+07 Memory= 1821.921 t=31 R=5e+05

Depth= 6116971 States= 1.5e+07 Transitions= 5.55e+07 Memory= 1909.616 t=33.6 R=4e+05

Depth= 6414843 States= 1.6e+07 Transitions= 5.93e+07 Memory= 1997.898 t=36.2 R=4e+05

Depth= 6642915 States= 1.7e+07 Transitions= 6.31e+07 Memory= 2086.667 t=38.9 R=4e+05

Depth= 6670587 States= 1.8e+07 Transitions= 7.6e+07 Memory= 2174.655 t=48.9 R=4e+05

Depth= 6670587 States= 1.9e+07 Transitions= 8.25e+07 Memory= 2266.257 t=53.4 R=4e+05

Depth= 6670587 States= 2e+07 Transitions= 8.96e+07 Memory= 2357.859 t=58.3 R=3e+05

Depth= 6670587 States= 2.1e+07 Transitions= 9.68e+07 Memory= 2449.558 t=63.5 R=3e+05

Depth= 6670587 States= 2.2e+07 Transitions= 1.04e+08 Memory= 2541.159 t=68.8 R=3e+05

Depth= 6670587 States= 2.3e+07 Transitions= 1.11e+08 Memory= 2632.761 t=74.4 R=3e+05

Depth= 6670587 States= 2.4e+07 Transitions= 1.19e+08 Memory= 2724.362 t=79.9 R=3e+05

Depth= 6670587 States= 2.5e+07 Transitions= 1.26e+08 Memory= 2815.964 t=85.4 R=3e+05

Depth= 6670587 States= 2.6e+07 Transitions= 1.33e+08 Memory= 2907.566 t=90.9 R=3e+05

Depth= 6670587 States= 2.7e+07 Transitions= 1.41e+08 Memory= 2999.167 t=96.6 R=3e+05

Depth= 6670587 States= 2.8e+07 Transitions= 1.48e+08 Memory= 3090.769 t=102 R=3e+05

Depth= 6670587 States= 2.9e+07 Transitions= 1.55e+08 Memory= 3182.370 t=108 R=3e+05

Depth= 6670587 States= 3e+07 Transitions= 1.63e+08 Memory= 3273.972 t=114 R=3e+05

Depth= 6670587 States= 3.1e+07 Transitions= 1.7e+08 Memory= 3365.573 t=120 R=3e+05

Depth= 6670587 States= 3.2e+07 Transitions= 1.77e+08 Memory= 3457.273 t=126 R=3e+05

(Spin Version 6.2.5 -- 3 May 2013)

+ Partial Order Reduction

Full statespace search for:

never claim + (p2\_1)

assertion violations + (if within scope of claim)

acceptance cycles + (fairness disabled)

invalid end states - (disabled by never claim)

State-vector 148 byte, depth reached 6670587, errors: 0

31367944 states, stored (3.28579e+07 visited)

1.594091e+08 states, matched

1.9226703e+08 transitions (= visited+matched)

26 atomic steps

hash conflicts: 1.014425e+08 (resolved)

Stats on memory usage (in Megabytes):

5265.005 equivalent memory usage for states (stored\*(State-vector + overhead))

2875.560 actual memory usage for states (compression: 54.62%)

state-vector as stored = 68 byte + 28 byte overhead

128.000 memory used for hash table (-w24)

534.058 memory used for DFS stack (-m10000000)

1.829 memory lost to fragmentation

3535.788 total actual memory usage

pan: elapsed time 139 seconds

No errors found -- did you verify all claims?

# Выводы

В рамках данной курсовой работы были исследованы подходы к верификации распределенных программ. Для модельной задачи были определены LTL формулы, описывающие корректное поведение системы. Далее модель была описана на языке Promela в системе Spin. После чего была проведена верификация с учетом описанных в ответе LTL формул. По результатам верификации можно судить о корректности модели.

Таким образом можно говорить об успешном выполнении задания и освоении инструментов верификации. Данные знания являются актуальными и могут применяться в большом количестве практик современного программирования.

# Список литературы

1. Ю.Г. Карпов Новая жизнь верификации: Издательство Открытые системы, 2012-03
2. Ю.Г. Карпов, И.В. Шошмина Верификация распределенных систем – СПб.: Издательство Политехнического университета, 2011.
3. Thomas Wahl – Fairness and Liveness URL: [http://www.ccs.neu.edu/home/wahl/Publications/fairness.pdf](http://www.ccs.neu.edu/home/wahl/Publications/fairness.pdf%20)
4. Concurrent programming lab2 URL:

<http://www2.compute.dtu.dk/courses/02158/sol_cplab2.html>

1. Andrew Ireland - Distributed Systems Programming (F21DS1) SPIN: Formal Analysis I URL: <http://www.macs.hw.ac.uk/~air/dsp-spin/lectures/lec-6-spin-2.pdf>
2. AG-Wehrheim – Verification with SPIN URL:

<http://www.cs.uni-paderborn.de/fileadmin/Informatik/AG-Wehrheim/Lehre/SS09/Model_Checking/Slides/11May09.pdf>