Государственное образовательное учреждение
высшего профессионального образования
(государственный университет)
«Московский государственный университет
имени М. В. Ломоносова»

Механико-математический факультет

Кафедра математической логики и теории
алгоритмов

О формализации модальной логики в системе Coq

On formalization of modal logic in the Coq Proof Assistant

Выпускная квалификационная работа
студентки 2 курса магистратуры
Красненковой Анастасии Владимировны

Научный руководитель
Кандидат физико-математических наук, доцент
Крупский Владимир Николаевич

Москва
2019

Содержание

1 Введение

В дипломной работе будет проанализирован вопрос доказательства так называемого "парадокса грязных детей"в системе Coq, а также верификация указанного доказательства средствами этой же системы.

В Приложениях 1 и 2 приведены программные коды, реализующие как доказательство парадокса, так и проводящие его верификацию.

Для успешной реализации поставленной задачи нам понадобится полимодальная эпистемическая система $S5^n$. Ее формулировка, семантика, натуральное исчисление и теорема корректности для него приведены в Приложении 1.

Приложение 2 полностью посвящено программной реализации "парадокса грязных детей".

Изложение в работе будет вестись максимально близко к методу построения программного кода.

Следуя традиции, будем использовать систему $S5^n$ для рассуждения о знании и его приращении.

2 Формализация модальной системы $S5^n$

В работе P. de Wind [1] строится натуральное исчисление для модальных систем T^n – $S5^n$ и приводится программная реализация данных исчислений в системе Coq. Мы подробно рассмотрим и модифицируем для наших нужд только систему $S5^n$ и дополним изложение необходимыми конструкциями, определениями и доказательствами.

Дадим определение формулы в указанной модальной системе.

Определение 1 $\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi\&\psi \mid \phi \vee \psi \mid \phi \to \psi \mid K_i\phi \mid C\phi$, где p – произвольная пропозициональная переменная.

В Coq синтаксис данных формул следующим образом:

```
Definition agent:=nat.
Inductive var:Set:=p:agent->var.
Inductive formula:Set:=
Top : formula
|Bottom : formula
```

```
|Atom : var->formula
|Not : formula->formula
|And : formula->formula->formula
|Or : formula->formula->formula
|Imp : formula->formula->formula
|K : agent->formula->formula
|C : formula->formula.
Infix  "-->"  := Imp (at level 8, right associativity).
Infix "||" := Or.
Infix "&&" := And.
```

Каждый полимодальный оператор $K_i, i \in A, A \subset N$ семантически ведет себя так же, как и обычный одномодальный оператор необходимости. (Каждый K_i – это, по сути, $\Box_i$), Множество агентов $A \subset N$ обычно предполагается конечным. Верхний индекс n в $S5^n$ как раз и обозначает мощность множества агентов A.

Формула $K_i\phi$ читается как "агент i знает, что ϕ".

Символ ' C ' — оператор общего знания.

Оператор C определяется следующим образом: построим оператор E_A, а затем с его помощью сконструируем искомый оператор общего знания.

Определение 2 $E_A\phi ::= \bigwedge\limits_{i \in A} K_i\phi$

Теперь введем следующую конструкцию ("степень оператора"):

Определение 3 $E_A^l\phi ::= E_A \circ E_A^{l-1}\phi$, $E_A^0\phi ::= \phi$.

Тогда

Определение 4 $C\phi ::= \forall l \in N\, E_A^l\phi$.

Заметим, что операторы K_i, как и оператор необходимости, а также степени данных операторов, обладают свойством пронесения через конъюнкцию $K_i^m(\phi\&\psi) \Longleftrightarrow (K_i^m\phi\&K_i^m\psi)$, m – степень оператора; причем в "степень"могут возводиться операторы с разными индексами, т.е. формула будет верна и для выражения с левой частью вида $K_iK_j(\phi\&\psi), i \neq j$

и т.п. Поэтому оператор общего знания C можно рассматривать как потенциально бесконечную конъюнкцию всех конечных последовательностей вида $K_iK_j.....\phi$ (или потенциально бесконечный набор всех конечных последовательностей указанного вида), все индексы принадлежат множеству A. Данное замечание будет полезно в дальнейшем, при осуществлении верификации натурального исчисления для S_5^n.

Также отметим, что так как в определении оператора C используются потенциально бесконечные конструкции, то его иногда определяют как неподвижную точку выражения $C_A\phi = \phi \& E_A(C_A\phi)$.

3 Семантическая интерпретация модальной системы $S5^n$

Зададим семантическую интерпретацию эпистемической системы $S5^n$.

Определение 5 *Пусть W – множество возможных миров, $A \subset N$ – конечное множество агентов, $R_i \subseteq W \times W, i \in A, 1 \leq i \leq n$ – набор отношений достижимости между мирами; каждое R_i рефлексивно ($\forall i \in A, \forall x \in W x R_i x$), симметрично ($\forall i \in A, \forall x, y \in W(xR_iy \rightarrow yR_ix)$) и транзитивно ($\forall i \in A, \forall x, y, z \in W((xR_iy \& yR_iz) \rightarrow xR_iz)$). Все точки множества W связаны между собой по какому-либо отношению R_i, т.е. у нас получается связный граф, каждое ребро которого принадлежит некоторому отношению R_i.*

Определение 6 *Отношение достижимости R^*, являющееся результатом рефлексивно-транзитивного замыкания по всем R_i, формально определим так: $R^* ::= (\bigcup_{i \in A} R_i)^*$.*

Заметим, что R^* является рефлексивным, симметричным и транзитивным отношением.

Определение 7 *R_U – универсальное отношение достижимости, т.е. $\forall x \in W \forall y \in W(xR_Uy)$, т.е. отношение R_U "видит"все точки из W.*

Лемма 1 *При условии связности W (см. Определение 5) R^* совпадает с R_U.*

Доказательство. По построению, рефлексивно-транзитивное замыкание отношения $\bigcup_{i \in A} R_i$ содержит само отношение $\bigcup_{i \in A} R_i$. По условию, каждая пара точек из W связана между собой по одному из отношений R_i, т.е. как раз по $\bigcup_{i \in A} R_i$. Как известно из теории графов, рефлексивность и симметричность отношения $\bigcup_{i \in A} R_i$ сохраняются при переходе к рефлексивно-транзитивному замыканию, а наличие транзитивных стрелок между любыми двумя точками из W следует из связности графа, построенного на основе W, по $\bigcup_{i \in A} R_i$ и определения транзитивного замыкания.

Обратные транзитивные стрелки получаются из симметричности.

Таким образом, R^* совпадает с R_U.

$\lhd$

Заметим, что ограничение интерпретаций логики $S5^n$ только связными графами сохраняет свойство полноты этой логики.

Для несвязных структур Лемма 1 неверна: возьмем, к примеру, в качестве шкалы несязную сумму универсальных кластеров. Очевидно, в такой шкале будет нарушаться универсальное свойство, но ее логикой также будет $S5$ ($S5^n$).

Дадим теперь строгие определения шкалы и модели для логики $S5^n$.

Обозначим множество всех $R_i, i \in A, 1 \leq i \leq n$ как R_A, каждое R_i рефлексивно, симметрично, транзитивно.

Свойства рефлексивности, транзитивности и симметричности постулируются явно.

Для универсального отношения R_U вводится специальная аксиома un: $\forall x, y \in W x R_U y$.

Тогда

Определение 8 *$F = (W, A, R_A, R_U, =, \neq)$ – шкала Крипке для $S5^n$, R_U – универсальное отношение. Отношения $=$, $\neq$ действуют на множестве агентов A естественным образом.*

Пусть $V : Var \to 2^W$ – функция оценки из множества всех переменных во множество всех подмножеств W. Тогда

Определение 9 *$M = (F, V)$ есть модель Крипке, основанная на шкале F.*

В Coq данная формализация выглядит так:

```
Record frame:Type:={
W : Set; (* worlds *)
R: agent->(W->(W->Prop));
RU: W->(W->Prop)}.
Record kripke: Type:={
F : frame; (* F=(W,R, RU) *)
L : (W F)->var->Prop}.

Axiom reff:  forall (M: kripke), forall i: agent,
forall y: (W (F M)),(R  (F M) i y y).

Axiom symm:  forall (M: kripke), forall i: agent,
forall x y: (W (F M)),(R (F M) i x y  -> R (F M) i y x).

Axiom trans:  forall (M: kripke), forall i: agent,
forall x y z: (W (F M)),((R (F M) i x y /\ R (F M) i y z) -> R (F M) i x z ).

Axiom un:  forall (M: kripke),  forall x y : (W (F M)),(RU(F M)  x y ).
```

Определим истинность произвольной формулы в точке x модели M (истинность формулы в отмеченной модели Крипке – отмечены точка носителя и агент; агент отмечается в тех формулах, где это необходимо).

Определение 10 $M, x \nvDash \bot$
$M, x \vDash \top$
$M, x \vDash p \rightleftharpoons x \in V(p)$
$M, x \vDash \neg\phi \rightleftharpoons M, x \nvDash \phi$
$M, x \vDash \phi\&\psi \rightleftharpoons (M, x \vDash \phi) \wedge (M, x \vDash \psi)$
$M, x \vDash \phi \vee \psi \rightleftharpoons (M, x \vDash \phi) \vee (M, x \vDash \psi)$
$M, x \vDash \phi \to \psi \rightleftharpoons (M, x \vDash \phi \Longrightarrow M, x \vDash \psi)$
$M, x \vDash K_i\phi \rightleftharpoons \forall y(xR_iy \Longrightarrow M, y \vDash \phi)$, i – произольный, $1 \leq i \leq n$.
$M, x \vDash C\phi \rightleftharpoons \forall y(xR_Uy \Longrightarrow M, y \vDash \phi)$

В Coq данное определение истинности формализуется так:

```
Fixpoint satisfies (M: kripke)(x :(W (F M))) (phi:formula) :Prop:=
match phi with
```

```
| Top => True
| Bottom => False
| (Atom phi)  => (L M x phi)
| (Not phi) => ~(satisfies M x  phi)
| (And phi_1 phi_2) =>
(satisfies M x  phi_1) /\ (satisfies M x  phi_2)
|(Or phi_1 phi_2) =>
(satisfies M x  phi_1) \/ (satisfies M x  phi_2)
|(Imp phi_1 phi_2) =>
(satisfies M x  phi_1) -> (satisfies M x  phi_2)
|(K i phi) =>
(forall y:(W (F M)),(R (F M) i x y) -> (satisfies M y  phi))
|(C  phi) =>
(forall y:(W (F M)), (RU (F M)  x y) -> (satisfies M y  phi))
end.
```

Определение истинности для формулы $C\phi$ корректно, т.е. оно действительно задает семантическую интерпретацию оператора общего знания C – см. [3].

Покажем корректность указанного определения истинности для оператора C явно. Заметим, что приводимое рассуждение во многом опирается на доказательство, построенное в Coq (см. теоремы reflax и pseudotrans).

Напомним, что $M, x \vDash C\phi \rightleftharpoons M, x \vDash E_A^k\phi$ для всех $k \in N$.

Обозначим последний пункт $M, x \vDash C\phi \rightleftharpoons \forall y(xR_Uy \Longrightarrow M, y \vDash \phi)$ из Определения 10 за $*$.

Теорема 1 *Определение истинности для формул вида $C\phi$ корректно для класса моделей с универсальным свойством, т.е. истинность $C\phi$ в смысле определения 4 эквивалентна истинности $C\phi$ в смысле $*$.*

Доказательство

Докажем необходимость.

Предварительно зафиксируем произвольную модель M.

Пусть $C\phi$ истинно в точке x в смысле определения 4, и xR_Uy. Рассмотрим путь из точки x в точку y и соответствующую ему последовательность операторов $\alpha\phi := K_iK_j...\phi$. Такой путь существует, т.к. граф связен, а отношения R_i симметричны.

Согласно определению 4, $\alpha\phi$ истинно в мире x, поэтому ϕ истинна в мире y.

Докажем достаточность.

Пусть $C\phi$ истинна в точке x модели M в смысле $*$.

Если $C\phi$ истинна в мире x в смысле $*$, то ϕ истинна во всех мирах модели. Тогда во всех мирах истинна формула $K_j\phi$, откуда следует истиность $K_iK_j\phi$ во всех мирах, и т.д. Тем самым, для каждой конечной последовательности $\alpha\phi$, составленной из операторов $K_i, i \in A$, установлена истинность $\alpha\phi$ во всех мирах модели. Тем самым, $C\phi$ истинна в смысле определения 4 в мире x (и во всех других мирах тоже).

$\lhd$

Введем следующие определения. Ниже каждого определения будет указана его формализация в Coq.

Определение 11 *Формула ϕ истинна в модели $M \Longleftrightarrow$ она истинна в каждой ее точке*

```
Definition M_satisfies M phi  := forall w:(W (F M)), (satisfies M w  phi).
```

Определение 12 *Формула ϕ истинна в шкале $F \Longleftrightarrow$ она истинна в каждой модели M, основанной на этой шкале.*

```
Definition valid_in_frame F phi :=
forall L, (M_satisfies (Build_kripke F L) phi).
```

Определение 13 *Формула ϕ общезначима в классе шкал $C \Longleftrightarrow$ она истинна в каждой шкале из данного класса.*

В Coq явно не вводится понятие класса шкал, но в коде накладываются ограничения на отношения достижимости, и эти ограничения и определяют класс шкал: каждое R_i рефлексивно, симметрично, транзитивно; есть универсальное отношение R_U, по которому из каждой точки видно каждую точку (см. код под Определением 9).

Приведем код, формализующий понятие общезначимой формулы.

```
(* |= phi *)
Definition valid phi:= forall F, (valid_in_frame F phi).
```

Мы рассматриваем только класс шкал из Определения 8, т.к. в Coq на отношения достижимости уже наложены соответствующие ограничения (см.выше).

Определим теперь понятие логического следования формулы ϕ из множества формул Γ.

Определение 14 *Формула ϕ логически следует из множества формул Γ в модели $M \Longleftrightarrow \forall w \in W(\forall \psi \in \Gamma(M, w \vDash \psi) \Longrightarrow (M, w \vDash \phi))$.*

Код будет таким:

```
Definition Satisfies F L Γ w := forall phi,
In phi  Γ -> satisfies (Build_kripke F L) w phi.

Definition Entailment F Γ L psi :=   forall w,

(Satisfies F L Γ w  -> satisfies (Build_kripke F L) w psi).
```

Определение 15 *Формула ϕ логически (глобально) следует из множества формул Γ в шкале $F \Longleftrightarrow \phi$ логически следует из множества формул Γ в каждой модели M, основанной на этой шкале.*

```
Definition Entailment1 Γ phi := forall F,forall L,  Entailment F Γ L phi.
```

Определение 16 *Формула ϕ логически следует из множества формул Γ классе шкал $C \Longleftrightarrow \phi$ логически следует из множества формул Γ в каждой шкале из данного класса.*

4 Синтаксическая реализация модальной системы $S5^n$

Синтаксическая реализация будет осуществляться через систему натурального вывода для $S5^n$.

Зададим данную систему следующим образом.

Пусть $l \in W$ – это возможный мир, или уровень (в терминологии P. de Wind).

В Coq данное понятие (уровень) формализуется так:

```
Inductive label : Set :=
o : label | k : agent->label | c: label.
(* k and c
are labels for the modal
connectives K and C *)
Inductive level : Set :=
nil : level |
cons : label -> level -> level.
```

Мир (уровень), полученный в результате перехода к некоторому "свежему"возможному миру, достижимому из l, обозначим как $Increase(l, K_i)$ или $Increase(l, C)$. Второй аргумент функции $Increase(x, y)$ (он же метка, label) зависит от того, по какому отношению достижимости был совершен переход: по какому-либо из R_i для соответствующего K_i или же по универсальному отношению достижимости R_U для C.

Будем использовать в данном разделе понятия "мир"и "уровень"как синонимы.

Например, утверждение "мир z достижим по R_i из мира y, который, свою очередь, достижим по R_i из мира l"запишется как

$Increase(Increase(l, K_i), K_i), y = Increase(l, K_i), z = Increase(Increase(l, K_i), K_i)$ и т.п.

В Coq переход к уровню (миру) $z, z = Increase(Increase(l, K_i), K_i)$ с уровня l будет выглядеть как список

```
[ki  ki  l]
```

Также нам понадобится переход к предшественнику возможного мира l, осуществляемый по обратным отношениям $R_i^{-1} = R_i$ и $C^{-1} = C$ (равенства корректны в силу симметричности отношений достижимости). Обозначим мир, полученный в результате данного перехода, как $Decrease(l)$. Аналогично функции Increase, возможен переход к предшественнику предшественника (мир $Decrease(Decrease(l))$) и т.п.

Заметим, что второй аргумент функции $Decrease(x)$ отсутствует по техническим причинам. Отслеживание, по какому именно отношению достижимости произошел переход к уровню-предшественнику, будет проводиться с помощью функций $Check_k$ и $Check_c$ (см.ниже).

Переход к предшественнику мира z из предыдущего примера (т.е. к миру y) будет выглядеть в Coq как список

```
[o  ki  l]
```

(стираем первый символ в строке).

Уровни (аналоги миров) представлены в Coq как списки, позволяющие отслеживать достижимость между мирами (уровнями).

Функции Increase (x,y) и Decrease(x), а также ряд вспомогательных функций, необходимых для корректной работы программы, реализованы в Coq.

```
Definition Increase (l:level) (lab: label) :level :=(cons lab l).
(* add a label in front of the list *)

Fixpoint Decrease (l:level): level:=
match l with
|nil => nil|
(cons o l') => (cons o (Decrease l'))|
(cons (k i) l') => (cons o l')|
(cons c l') => (cons o l')
end.
```

Вспомогательная функция $Remove_o(l)$ стирает нули слева, т.к. они лишние и не несут никакой полезной информации о достижимости. Нули появляются после работы функции $Decrease$.

```
Fixpoint Remove_o (l: level): level :=
match l with
|nil => nil |
(cons e l') => match e with
|o => (Remove_o l') |
(k i) => (cons e l')|
c => (cons e l')
end
end.
(* remove all o's at the front of the list *)
```

Для описания правил вывода нам понадобятся функции $Check_k(l, i)$ и $Check_c(l)$. Область значений обеих функций – $\{\top, \bot\}$, l – мир-потомок, i – агент. Первая функция проверяет, был ли осуществлен переход между мирами (уровнями) по отношению R_i, а вторая – по R_U.

```
Fixpoint Check_c (l:level): Prop:=
match l with
nil => False |
(cons o l') => (Check_c l') |
(cons (k i) l') => False |
(cons c l') => True
end.
```

```
Fixpoint Check_k(i:agent) (l:level): Prop:=
match l with
nil => False |
(cons o l') => (Check_k i l') |
(cons c l') => False |
(cons (k a) l') => i=a
end.


Definition EqLevel (l l': level) :=(Remove_o l)=(Remove_o l').
(* two lists are equivalent if they are the same after
removing all o's at the front of the lists *)
```

Опреределение EqLevel (l l': level) утверждает, что миры (уровни), отличающиеся только нулями слева, равны.

Введем теперь параметр $Ass\, n\, \phi$, означающий, что на уровне (в мире) n в качестве допущения вводится формула ϕ.

```
Parameter Ass:level->formula->Prop.
```

Данная конструкция играет важную техническую роль и позволяет вводить в вывод в качестве гипотез необходимые для доказательства допущения.

Зададим теперь правила вывода нашей системы через индуктивное определение предиката $Provable\, n\, \phi$, обозначающего следующее: в мире (на уровне) n доказуема формула ϕ. Математически запишем это как $n \vdash \phi$.

Тогда для любых формул $\psi, \phi, \phi 1$, любых допущений, любого возможного мира (уровня) n, любого агента $i \in A$ верны следующие утверждения о выводимости (все данные утверждения и будут правилами вывода в нашей системе):

1. $\forall n,\ n \vdash \top$.

2. $\forall n \forall \phi (Ass\, n\, \phi \Longrightarrow n \vdash \phi)$ – правило 2 постулируется как аксиома

```
Axiom Prov: forall phi:formula, forall n:level,(Ass n phi)-> (Provable  n  phi)
```

.

2'. $\forall n, l\, \forall \phi (Eqlevel(n, l) \to (l \vdash \phi \to n \vdash \psi))$ – правило 2' говорит, что для эквивалентных уровней утверждения о доказуемости произвольной формулы также эквивалентны.

3. $\forall n \forall \phi \forall \psi (n \vdash \phi \Longrightarrow (n \vdash \psi \Longrightarrow (n \vdash (\phi \& \psi))))$ – это правило введения конъюнкции $\&_i$.

4. $\forall n \forall \phi \forall \psi (n \vdash \phi \& \psi \Longrightarrow (n \vdash \psi))$ – это правило удаления конъюнкции $\&_e 1$.

5. $\forall n \forall \phi \forall \psi (n \vdash \phi \& \psi \Longrightarrow (n \vdash \phi))$ – это правило удаления конъюнкции $\&_e 2$.

6. $\forall n \forall \phi \forall \psi (n \vdash \phi \Longrightarrow (n \vdash \phi \vee \psi))$ – это правило введения дизъюнкции $\vee_i 1$.

7. $\forall n \forall \phi \forall \psi (n \vdash \psi \Longrightarrow (n \vdash \phi \vee \psi))$ – это правило введения дизъюнкции $\vee_i 2$.

8. $\forall n \forall \phi (n \vdash \neg\neg\phi \Longrightarrow (n \vdash \phi))$ – это правило удаления двойного отрицания $\neg\neg_e$.

9. $\forall n \forall \phi \forall \psi (n \vdash \psi \to \phi \Longrightarrow (n \vdash \psi \Longrightarrow (n \vdash \phi)))$ – это правило удаления импликации $\to_e$.

10. $\forall n \forall \phi (n \vdash \phi \Longrightarrow (n \vdash \neg\phi \Longrightarrow (n \vdash \bot)))$ – это правило удаления отрицания $\neg_e$.

11. $\forall n \forall \phi (n \vdash \bot \Longrightarrow (n \vdash \phi))$ – это правило удаления противречия $\bot_e$.

Правила 1 – 11 — это правила, не требующие введения новых допущений.

12. $\forall n \forall \phi \forall \psi (((Ass\ n\, phi) \Longrightarrow n \vdash \psi) \Longrightarrow (n \vdash \phi \to \psi))$ – это правило введения импликации $\to_i$.

Данное правило утверждает следующее: если из допущения ϕ выводится формула ψ, то тогда выводится импликация $\phi \to \psi$ (справедливо для любого уровня).

13. $\forall n \forall \phi \forall \psi \forall \psi_1 (n \vdash (\phi \vee \psi) \Longrightarrow n \vdash (\phi \to \psi_1) \Longrightarrow (n \vdash \psi \to \psi_1) \Longrightarrow (n \vdash \psi_1))$

– правило удаления дизъюнкции $\vee_e$, действующее аналогично правилу введения импликации. Новые допущения здесь появляются через работу с импликацией (в качестве допущений вводим ϕ и ψ).

14. $\forall n \forall \phi (n \vdash \phi \to \bot \Longrightarrow (n \vdash \neg\phi))$ – это правило введения отрицания $\neg_i$. По аналогии с правилом 13, новые допущения появляются при работе с импликацией (допускаем ϕ).

Таким образом, правила 12 – 14 — это правила, требующие введения новых допущений.

15. $\forall n \in W \forall i \in A \forall \phi(Check(n, K_i) = \top \Longrightarrow (Decrease(n) \vdash K_i\phi \Longrightarrow (n \vdash \phi)))$ – это правило удаления K_i K_iE.

Сначала осуществляется проверка, получены ли уровни "предшественник"и "потомок"с помощью подходящего K_i. Если да, то необходимо, чтобы формула $K_i\phi$ выводилась на уровне (в мире), строго предшествующем n (выводилась в мире – непосредственном потомке).

Заметим, что в терминологии возможных миров данное правило не требует введения нового, "свежего"мира, ранее не встречавшегося в выводе, напротив, оно пользуется "свежим"миром, уже введенным по правилу K_iI.

16. $\forall n \forall i \in A \forall \phi(Increase(n, K_i) \vdash \phi \Longrightarrow (n \vdash K_i\phi))$ – это правило введения K_i K_iI.

Уровень, или аналог возможного мира, $Increase(n, K_i)$ – это свежий, ранее не встречавшийся в выводе возможный мир, достижимый из n по соответствующему R_i.

В построенной нами интерпретации формула ϕ должна выводиться строго на уровне

```
[ k i  n ]
```

, где n – уровень, на котором выводится $K_i\phi$.

17. $\forall n \forall i \in A \forall \phi(Increase(n, C) \vdash \phi \Longrightarrow (n \vdash C\phi))$ – это правило введения C C_i.

Правило работает аналогично правилу 16, только рассматривается отношение достижимости R_U.

18. $\forall \Gamma \forall n \in W \forall i \in A \forall \phi(Check(n, C) = \top \Longrightarrow (Decrease(n) \vdash C\phi \Longrightarrow (n \vdash \phi)))$ – это правило удаления C C_e.

Правило полностью аналогично правилу 15, вместо отношения R_i рассматривается отношение R_U.

19. $\forall n \forall i \in A \forall \phi(n \vdash C\phi \Longrightarrow (n \vdash K_i\phi))$ – это правило получения из C произвольного K_i CK.

Заметим, что это правило – часть Теоремы 1.

20. $\forall n \forall i \in A \forall \phi(n \vdash K_i\phi \Longrightarrow (n \vdash \phi))$ – это правило рефлексивности T.

21. $\forall n \forall i \in A \forall \phi(n \vdash K_i\phi \Longrightarrow (n \vdash K_iK_i\phi))$ – это правило транзитивности $K4$.

22. $\forall n \forall i \in A \forall \phi(n \vdash \neg K_i\phi \Longrightarrow (n \vdash K_i\neg(K_i\phi)))$ – это правило т.н. негативной интроспекции $K5$ (если агент не знает чего-то, то он знает о

своем незнании): для его верификации нужны симметричность и транзитивность K_i.

23. $\forall n \forall i \in A \forall \phi (n \vdash C\phi \Longrightarrow (n \vdash CC\phi))$ – это правило транзитивности C.

Правила 15 – 23 — это правила вывода, оперирующие с модальностями.

В Coq предикат доказуемости описывается так:

```
Inductive Provable: level->formula->Prop:=
|truth : forall n:level,(Provable n Top)
|ass_at :  forall phi:formula, forall l m:level,
(EqLevel l m) -> ((Provable l phi) -> (Provable  m phi))
|andI :  forall phi1 psi2:formula, forall n:level,
(Provable n phi1) -> (Provable  n psi2) ->
(Provable  n (And phi1 psi2))
|andE1 :  forall phi1 phi2:formula, forall n:level,
(Provable n (And phi1 phi2)) -> (Provable n phi1)
|andE2 : forall phi1 phi2:formula, forall n:level,
(Provable  n (And phi1 phi2)) -> (Provable  n phi2)
|orI1 :  forall phi1 phi2:formula, forall n:level,
(Provable  n phi1) -> (Provable n (Or phi1 phi2))
|orI2 :  forall phi1 phi2:formula, forall n:level,
(Provable n phi2) -> (Provable n (Or phi1 phi2))
|orE :  forall phi1 phi2 psi:formula, forall n:level,
(Provable  n (Or phi1 phi2)) ->
(Provable  n (phi1 --> psi)) ->
(Provable n  (phi2 --> psi)) -> (Provable  n psi)
|impI : forall phi1 phi2:formula, forall n:level,
((Ass n phi1) -> (Provable n phi2))->
(Provable  n (Imp phi1 phi2))
|impE :  forall phi1 phi2:formula,forall n:level,
(Provable  n (Imp phi1 phi2)) ->
(Provable  n phi1) -> (Provable  n phi2)
|notI : forall phi:formula, forall n:level,
(Provable n (phi --> Bottom)) ->
(Provable  n (Not phi))
|notE :  forall psi:formula, forall n:level,
(Provable  n psi) ->
```

```
((Provable  n (Not psi)) -> (Provable  n Bottom))
|botE : forall phi:formula, forall n:level,
(Provable  n Bottom) -> (Provable  n phi)
|notnotE :  forall phi:formula, forall n:level,
(Provable  n (Not (Not phi))) -> (Provable  n phi)
|KiE :  forall phi:formula, forall i: agent, forall n:level,
((Check_k i n) -> (Provable (Decrease n) (K i phi)) -> (Provable n phi))
|KiI :  forall phi:formula, forall i: agent, forall n:level,
(Provable  (Increase n(k i)) phi) -> (Provable  n (K i phi))
|CK:  forall phi:formula, forall i: agent, forall n:level, (*данное правило утв
(Provable  n (C phi)) -> (Provable  n (K i phi))
|CE :  forall phi:formula, forall i: agent, forall n:level,
((Check_c n) -> (Provable  (Decrease n) (C phi)) -> (Provable  n phi))
|CI :  forall phi:formula, forall i: agent,forall n:level,
(Provable (Increase n c) phi) -> (Provable  n (C phi))
|t :  forall phi:formula, forall i: agent, forall n:level,
(Provable  n (K i phi)) -> (Provable n phi)
|four :  forall phi:formula, forall i: agent, forall n:level,
(Provable  n (K i phi)) -> (Provable  n (K i (K i phi)))
|five :  forall phi:formula, forall n:level, forall i: agent,
(Provable  n (Not (K i phi))) ->
(Provable  n (K i (Not (K i  phi))))
|Cf :  forall phi:formula, forall i: agent, forall n:level,
(Provable  n (C phi)) -> (Provable  n (C (C phi))).
```

Каждая формула в выводе есть либо допущение, либо получена из предыдущих по одному из описанных выше правил.

Определение 17 *Назовем формулу ϕ выводимой в системе S_5^n, если она выводима из пустого множества допущений и в любом возможном мире.*

В терминологии уровней это означает, что формула выводима на пустом уровне (ее выводимость не зависит от условий на достижимость, уровней-предшественников и потомков), т.е. $Provable \epsilon \phi$, где ϵ – пустой уровень.

Приведем примеры формул, выводимых в S_5^n.

Оба примера доказаны в Coq (леммы propDis и mod5).

Пример 1 *Покажем, что* $\epsilon \vdash_{S_5^n} (p \vee q) \to (q \vee p)$ *(моделируем рассуждение в Coq).*

Зависимость формул от допущений будем помечать справа с помощью номера соответствующего допущения в квадратных скобках, например, [3].

Допущение зависит от самого себя.

1 .Ass $\epsilon(p \vee q)$ *– допущение, см.правило* to_i.

2. $\epsilon \vdash_{S_5^n} p \vee q$ *– правило 2: 1 [1]*

3. Ass $\epsilon\, p$ *– допущение, см.правило* $\vee_e$.

4. $\epsilon \vdash_{S_5^n} p$ *– правило 2: 3 [3]*

5. $p, (p \vee q), \epsilon \vdash_{S_5^n} q \vee p$ *–* $\vee_i 2$*: 4 [1, 3]*

6. Ass $\epsilon\, q$ *– допущение, см. правило* $\vee_e$.

7. $\epsilon \vdash_{S_5^n} q$ *– правило 2: 6 [6]*

8 .$q, (p \vee q), \epsilon \vdash_{S_5^n} q \vee p$ *–* $\vee_i 1$*: 7 [1, 6]*

9. $(p \vee q), \epsilon \vdash_{S_5^n} q \vee p$ *–* $\vee_e$*: 2, 5, 8 [1]*

10. $\epsilon \vdash_{S_5^n} (p \vee q) \to (q \vee p)$ *–* $\to_i$*: 6 [∅]*

Формула доказуема на пустом уровне и из пустого множества допущений – значит, она доказуема повсеместно, на любом уровне или в любом возможном мире.

Пример 2 *Покажем, что* $\epsilon \vdash_{S_5^n} K_i(p \to q) \to (K_i p \to K_i q)$ *(моделируем рассуждение в Coq).*

1.Ass $\epsilon\, K_i(p \to q)$ *– допущение, см.правило* to_i.

2. $\epsilon \vdash_{S_5^n} K_i(p \to q)$ *– правило 2: 1 [1]*

3. Ass $\epsilon\, K_i p$ *– допущение, см.правило* to_i.

4. $\epsilon \vdash_{S_5^n} K_i p$ *– правило 2: 3 [3]*

5. $Decrease(Increase(\epsilon\,, K_i)) = \epsilon$ *– по определению Increase и Decrease.*

6. $Check(\epsilon\,, K_i) = \top$ *– по условию.*

7. $K_i(p \to q), K_i p, Increase(\epsilon\,, K_i) \vdash_{S_5^n} p \to q$ *–* $K_i E$*: 5, 6, 2 [1]*

8. $K_i(p \to q), K_i p, Increase(\epsilon\,, K_i) \vdash_{S_5^n} p$ *–* $K_i E$*: 5, 6, 4 [3]*

9. $K_i(p \to q), K_i p, Increase(\epsilon\,, K_i) \vdash_{S_5^n} q$ *–* $\to_e$*: 5, 6, 4 [1, 3]*

10. $K_i(p \to q), K_i p, Decrease(Increase(\epsilon\,, K_i)) = \epsilon \vdash_{S_5^n} K_i q$ *–* $K_i I$*: 5, 9 [1,3]*

Здесь мы видим замкнутость множества допущений по K_i*, что соответствует правилам секвенциальной логики.*

11. $K_i(p \to q), \epsilon \vdash_{S_5^n} K_i p \to K_i q$ *–* $\to_i$*: 10 [1]*

12. $\epsilon \vdash_{S_5^n} K_i(p \to q) \to (K_i p \to K_i q)$ *–* $\to_i$*: 11 [∅]*

Все допущения вводятся согласно соответствующим правилам вывода и потом устраняются с помощью этих же правил. Поэтому доказуемая формула от допущений не зависит.

В Coq мною была доказана теорема корректности: $Prop\,Soundness$ $\forall\phi(\epsilon \vdash \phi \Longrightarrow \vDash \phi)$.

В формулировке теоремы корректности общезначимость определяется по отношению к введенному ранее классу шкал.

Доказательство идет индукцией по длине вывода, последовательно разбираются случаи применения каждого из указанных выше правил вывода (см.код). В случае применения некоторых модальных правил нам потребуется вложенная индукция по глубине уровня.

В Coq данная теорема формулируется следующим образом:

```
Definition Prop_Soundness :=  forall phi:formula,
Provable  nil  phi -> valid phi.

Theorem Soundness: Prop_Soundness.
```

Теорема 2 $\forall\phi(\epsilon \vdash \phi \Longrightarrow \vDash \phi)$

Доказательство

Для Правила 1 нужно доказать: $n \vdash \top \Longrightarrow \forall F \forall V \forall w \in W(M = (F, V), w \vDash \top)$, n – произвольный уровень.

Доказывается сразу, без индуктивной гипотезы, так как во всех моделях $S5^n$ общезначима истина $\top$. В Coq доказательство также проходит почти автоматически после раскрытия определения общезначимости и истинности в шкалах и моделях.

Правило 2 для допущений не верифицируем, оно идет как аксиома, но для работы с правилами, которые требуют введения новых допущений, мы используем такой постулат:

```
Axiom IAss: forall phi psi:formula, forall F, forall L,
forall l: level, forall w,
(((Ass l phi) -> satisfies (Build_kripke F L) w psi)
/\ satisfies (Build_kripke F L) w phi)
-> satisfies (Build_kripke F L) w psi.
```

Данный постулат позволяет учитывать в процессе верификации информацию об истинности формул-допущений. Его смысл прост: если формула является допущением и из нее следует истинность некоторой формулы в произвольной модели в произвольном мире w, и при этом формула-допущение является истинной в той же модели и в том же мире, то формула, истинность которой следует из допущения, действительно является истинной в рассматриваемой модели и указанном мире w.

Правило 2'. Получается автоматически, с помощью одной строки кода

```
auto.
```

, т.к. индуктивная гипотеза совпадает с целью.

Правило 3. $\forall n \forall \phi \forall \psi (n \vdash \phi \Longrightarrow (n \vdash \psi \Longrightarrow (n \vdash (\phi \& \psi))))$ – правило введения конъюнкции $\&_i$.

Здесь нам потребуются индуктивные гипотезы для посылок данного правила вывода:

$IH_1 : M, n \vDash \phi$, $M = (F, L)$, n – произвольный мир (уровень).

$IH_2 : M, n \vDash \psi$

Отсюда по определению истинности для модальных формул (Определение 10) сразу же следует $M, \Gamma, n \vDash \phi \& \psi$.

Рассуждение справедливо для любого уровня (возможного мира) n и любой модели M, в силу общезначимости индуктивных гипотез.

Следовательно, и заключение правила общезначимо.

Аналогичные рассуждения проходят в случаях 4 – 11, не будем останавливаться на них подробно, так как они ничем не отличаются от рассуждений в пропозициональной логике.

Рассмотрим правило введения имликации, которое требует добавления новых допущений (правило 12).

$\forall n \forall \phi \forall \psi (((Ass\, n\, \phi) \Longrightarrow n \vdash \psi) \Longrightarrow (n \vdash \phi \to \psi))$

Нам надо доказать, что $n \vDash \phi \to \psi$ для любого мира n. Тогда для произвольной модели $M = (F, L)$ и произвольного уровня (мира) n должно быть верно, что $M, n \vDash \phi \Longrightarrow M, n \vDash \psi$.

Введем гипотезу:

$H1 : M, n \vDash \phi$.

Введем также допущение – индуктивную гипотезу:

$H : M, (Ass\, n\, \phi) \Longrightarrow \vDash \psi$. В Coq это записывается как

```
(Ass n  phi) -> valid   psi
```

Надо доказать, что $M, n \vDash \psi$.

Используя постулат IAss и гипотезу H1, почти сразу же получаем требуемое – для завершения доказательства надо показать, что $M, n \vDash \phi$, но для этого у нас есть гипотеза $H1$.

Отметим, что Coq автоматически осуществляет переход от утверждения об истинности в модели к утверждению об истинности в той же модели, но в некотором ее мире n.

По похожей схеме идет доказательство правила удаления дизъюнкции – правила 13 .

Здесь используются три индуктивные гипотезы, соответствующие посылкам:

$IH_1 : M, n \vDash \phi \vee \psi \Longleftrightarrow (M, n \vDash \phi$ или $M, n \vDash \psi)$, $M = (F, L)$.

$IH_2 : M, n \vDash \phi \to \psi_1$.

$IH_3 : M, n \vDash \psi \to \psi_1$.

Надо доказать, что $M, n \vDash \psi_1$.

Элиминируя IH_1, получаем две новые гипотезы:

$H : M, n \vDash \phi$.

$H_0 : M, n \vDash \psi$.

Также нам нужно дважды доказать, что $M, n \vDash \psi_1$. Используя IH_1, IH_2, H, H_0, а также определение истинности импликации в точке, получаем требуемое.

Правило 14 разбирается очень просто, т.к. $\neg\phi$ эквивалентно в Coq формуле $\phi \to \bot$, являющейся посылкой данного правила.

Перейдем теперь к модальным правилам.

Правило 15. $\forall n \forall i \in A \forall \phi (Check(n, K_i) = \top \Longrightarrow (Decrease(n) \vdash K_i\phi \Longrightarrow (n \vdash \phi)))$.

Верификация данного правила требует вложенной индукции по глубине уровня. Заметим, что в силу построения уровня и действия функции Decrease обоснование индуктивного перехода идет в обратном направлении – от большего к меньшему (утверждение справедливо для большего уровня, надо доказать для меньшего).

Пусть наш уровнень n пустой. Тогда уровня $Decrease(\epsilon)$ не существует (пустой уровень ниоткуда не достижим). В этом случае все автоматически верно в силу ложности посылки $Decrease(n)$ (база индукции).

Пусть уровень n не пуст. Coq создает следующую индуктивную гипотезу:

```
IHn : Check_k i n -> Provable (Decrease n) (K i phi) -> valid phi
```

. Теперь, если агент $a <> i$, то в силу ложности посылки индуктивной гипотезы $Check_k(n,\ K_i)$, все верно.

Пусть $i = a$. Посылка $Check_k(n,\ K_i)$ верна. Тогда рассмотрим структуру произвольного уровня n, $n = label\, n'$. Для уровня n' все верно по предположению индукции . Здесь также возможно несколько случаев, в зависимости от типа метки label:

1. $n = on'$. Тогда, по сути, $n = n'$, и мы можем прямо воспользоваться предположением индукции для глубины уровня – поэтому применяем IHn. Учитывая, что уровни n и on' эквивалентны (правило Eqlevel в Coq), а также учитывая доказуемость Provable (Decrease n) (K i phi) , которая у нас есть по условию теоремы, получаем утверждение об общезначимости ϕ.

Код выглядит так:

```
apply IHn.
auto.
apply ass_at with (cons o (Decrease n)).
unfold Decrease.
unfold EqLevel.
unfold Remove_o.
reflexivity.
exact H0.
```

2. $n = ki\, n'$. Тогда, используя функцию Decrease, получаем, что Provable (o n') (K i phi), или упрощая, Provable n' (K i phi). Так как уровень n' меньшей глубины, чем n, то применяем предположение индукции для глубины вывода (не уровня!), т.е.

$IH : M, n' \vDash K_i\phi$. В программе

```
IHProvable : valid (K i phi)
```

Надо доказать, что $n' \vDash \phi$.

Здесь нам потребуется рефлексивность отношения R_i (или формула рефлексивности, доказанная в Coq как reflax).

Применяя общезначимую для нашего класса шкал формулу рефлексивности $K_i\phi \to \phi$, получаем $n' \vDash \phi$.

Отметим, что в случае $n = on'$ формула $K_i\phi$, выводимая на уровне $Decrease(n)$, не общезначима, но используется в выводе общезначимой формулы, частью которой является ϕ (поэтому Coq и пишет: valid phi).

Таким образом, при соответствии определенным условиям, о которых речь была выше, истинность в фиксированной при верификации произвольной модели, а следовательно, и корректность сохраняется.

Проще говоря, мы проверяем здесь наличие логического следования из множества формул Г, частью которого является формула $K_i\phi$ (см.определения семантического раздела), поэтому имеем право писать valid phi.

Если же эти условия не выполняются, то вывод возможен только в случае уже доказанной (!) общезначимой формулы ϕ. И тогда это верификация рефлексивности для K_i.

3. $n = cn'$. Посылка $Check_k(n, K_i)$ ложна, и все проходит автоматически.

Таким образом, данное правило проходит для любого уровня n.

Правило 16. $\forall n \forall i \in A \forall \phi(Increase(n, K_i) \vdash \phi \Longrightarrow (n \vdash K_i\phi))$.

Верификация данного правила требует вложенной индукции по глубине уровня. Заметим, что в силу построения уровня и функции Increase обоснование индуктивного перехода идет в прямом направлении – от меньшего уровня к большему.

Пусть наш уровнень n пустой. Рассмотрим уровень $Increase(\epsilon\,, K_i)$.

Здесь необходимо заметить, что любой уровень вида $Increase(n, K_i)$ получается из уровня n (по построению исчисления) только двумя способами: I. либо на уровне n в выводе верна формула KiF (или CF, из которой получается KiF по правилу СК), введенная как допущение, F – произвольная формула; либо II. формула K_iF является целью, и формул вида K_iG, G – произвольная формула, в выводе нет.

Тогда (в случае II) предполагается, что формула K_iF – тавтология, выводимая на пустом уровне ϵ. Следовательно, в выводе появляются допущения, находящиеся на уровне $Increase(\epsilon\,, K_i)$. Формула F, также по модальным правилам, должна быть тавтологией, выводимой на пустом уровне.

Отсюда следует, что верификация базы индукции для правила 16 сводится к верификации правила Nec для каждого K_i. Поэтому пользуемся индуктивной гипотезой по глубине вывода:

Итак, для произвольной модели $M = (F, L)$ и агента i верна индуктивная гипотеза:

$IH : M, \forall x \in W \vDash \phi$. В Coq это

```
IHProvable : valid   phi
```

.

Надо доказать, что $M, \forall x \in W \vDash K_i\phi$ для любых M, i . Фиксируем M, i и возможный мир x. Данное утверждение равносильно истинности следующего выражения $M, x \vDash \forall z(xR_iz \to \phi)$.

Вводим такие допущения:

$H1 : xR_iz$ $(z = Increase(\epsilon\,, K_i))$. Отметим еще раз, что выводимость на пустом уровне ϵ обзначает истинность во всех возможных мирах.

Надо доказать, что $M, z \vDash \phi$.

В IH снимаем квантор всеобщности, подставляем мир z, получаем $M, z \vDash \phi$.

Правило Nec (для каждого K_i) верифицировано, база индукции доказана.

Пусть уровень n не пуст.

Тогда имеет место случай I (на уровне n в выводе верна формула K_iF (CF), введенная как допущение). Уровень $Increase(n, K_i)$ получается из уровня n через снятие K_i с формулы F , формула F находится на уровне $Increase(n, K_i)$. Искомая формула ϕ выводится из формулы F тоже на уровне $Increase(n, K_i)$, т.к. использование допущений с других уровней запрещено привязкой предиката Provable к фиксированному уровню (возможному миру).

Поэтому, в случае успешной верификации, будет выполнено правило секвенциальной модальной логики Closure: $\Gamma \vdash \phi \Longrightarrow K_i\Gamma \vdash K_i\phi$ (Closure1: $\Gamma \vdash \phi \Longrightarrow C\Gamma \vdash K_i\phi$), гарантирующее корректность исчисления.

Множество допущений Γ эквивалентно формуле F, представляющей собой конъюнкцию всех формул из Γ.

Coq создает следующую индуктивную гипотезу:

```
IHn : Provable (Increase n (k i)) phi -> valid (K i phi)
```

.

Здесь следует отметить, что мы доказываем следующее: в случае I формула ϕ ,выводимая на уровне $Increase(K_i, n)$, не общезначима, но используется в выводе общезначимой формулы, частью которой является $K_i\phi$ (поэтому Coq и пишет valid (K i phi)). Таким образом, при соответствии определенным условиям, о которых речь пойдет ниже, истинность в фиксированной при верификации произвольной модели, а следовательно, и корректность сохраняется.

Проще говоря, мы в данном случае проверяем наличие логического следования из множества формул Γ, поэтому имеем право писать valid (K i phi).

Заметим, что в данном случае множество формул Γ должно быть модализировано, т.е. должно выполняться условие $\forall\psi \in \Gamma \exists\phi(\psi = Ki\phi \vee \psi = C\phi)$, для того, чтобы правило Closure/ Closure1 сработало.

Если же эти условия не выполняются, то вывод возможен только в случае уже доказанной (!) общезначимости формулы ϕ. И тогда это верификация Nec для K_i.

Вернемся к случаю непустого уровня. Анализируем его в зависимости от типа метки.

1. $n = on'$. Тогда, по сути, $n = n'$, и мы можем прямо воспользоваться предположением индукции для глубины уровня – поэтому применяем IHn. Учитывая, что уровни n и on' эквивалентны (правило Eqlevel в Coq), а также учитывая доказуемость Provable (Increase n (k i)) phi, которая у нас есть по условию теоремы, получаем утверждение об общезначимости $K_i\phi$.

Код выглядит так:

```
simpl in H.
simpl in IHn.
apply IHn.
apply ass_at with (Increase (cons o n) (k i)).
unfold EqLevel.
unfold Remove_o.
simpl.
subst.
assert (H0: cons o n = n).
apply Check.
symmetry in H0.
rewrite <- H0.
tauto.
exact H.
```

Здесь мы также используем технический и очень естественный постулат Check, утверждающий, что уровни, различающиеся только лидирующими нулями, равны .

```
Axiom Check: forall n: level, cons o n = n.
```

2. $n = kjn'$, $j \neq i$. Т.к. достижимость по R_i (нужна для обоснования истинности $K_i\phi$) не эквивалентна в общем случае достижимости по R_j, то утверждение

```
Provable (Increase (cons (k j) n) (k i)) phi
```

, проще говоря, выводимость, верная для любого уровня n такая, что $Provable\, Increase(K_i, (Increase(K_j, n))\, \phi$ (рассматривается уровень ⌊k i kj n⌋), справедливо только в том случае, когда ϕ – тавтология (уже на данный момент доказательства эквивалентна $\top$); поэтому формула ϕ свободно поднимается на произвольный уровень, по любому отношению достижимости.

В силу вышесказанного, верификация в данном случае представляет собой верификацию Nec для K_i.

3. $n = kin'$, $j = i$. В этом случае у нас есть утверждение

```
Provable (Increase (cons (k i) n) (k i)) phi
```

, проще говоря, выводимость, верная для любого уровня n такая, что $Provable\, Increase(K_i, (Increase(K_i, n))\, \phi$. (Рассматривается уровень ⌊k i ki n⌋). Так как ϕ в данном случае поднимается с любого произвольного уровня на любой другой произвольный уровень по отношению достижимости R_i, то она может быть только тавтологией. Иначе может нарушиться корректность, и докажется, например, необщезначимая формула вида $\phi \to K_i\phi$.

Поэтому верификация в данном случае представляет собой верификацию Nec для K_i.

4. $n = cn'$. Рассуждения полностью аналогичны случаям 2 и 3. Также происходит верификация Nec для K_i.

Правила 17 –18 рассматриваются совершенно аналогично, с той лишь разницей, что в правиле удаления C используется функция $Check_c(x)$.

Правило 19. $\forall n \forall i \in A \forall \phi (n \vdash C\phi \Longrightarrow (n \vdash K_i\phi))$.

Данное правило используется в Теореме 1.

Для произвольной модели $M = (F, L)$, произвольного уровня (мира) n и произвольного агента i верна индуктивная гипотеза:

$IH : M, n \vDash C\phi \Longleftrightarrow \forall y (nR_U y \Longrightarrow M, n \vDash \phi)$ (см.Определение 10).

Нам нужно доказать для произвольного фиксированного i, что:

$M, n \vDash K_i\phi \Longrightarrow \forall y (xR_i y \Longrightarrow M, y \vDash \phi)$.

Упрощая, приходим к тому, что нам требуется доказать:

$M, y \vDash \phi$, y фиксирован.

Применяя индуктивную гипотезу ("apply IH"в Coq), получаем, что нам нужно доказать $nR_U y$.

Для этого применяем аксиому un: $\forall x, yxR_U y$, характеризующую универсальное отношение достижимости, откуда сразу же следует требуемое.

Правило 20. $\forall n \forall i \in A \forall \phi (n \vdash K_i \phi \Longrightarrow (n \vdash \phi))$ – это правило рефлексивности T.

Уже было доказано ранее в Coq в качестве теоремы reflax и использовалось при верификации правил 15 и 18.

Рассмотрим данное доказательство в контексте теоремы корректности.

Для произвольной модели M, произвольного мира n и агента i верна индуктивная гипотеза:

$IH : M, n \vDash K_i \phi \Longleftrightarrow \forall y (lR_i y \Longrightarrow M, y \vDash \phi)$.

Нам нужно доказать, что:

$M, n \vDash \phi$.

Применим аксиому рефлексивности для произвольного отношения R_i (refl: $\forall i \in A \forall w \in W, wR_i w$, также верная для всех моделей нашего типа). Поэтому из нее и IH сразу же следует требуемое.

Правило 21. $\forall n \forall i \in A \forall \phi (n \vdash K_i \phi \Longrightarrow (n \vdash K_i K_i \phi))$.

Доказывается аналогично правилу 20, только с использованием аксиомы trans.

Код:

```
unfold valid. intros. unfold valid_in_frame. intros.
unfold M_satisfies. intros.
unfold valid in IHProvable.
unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
simpl in IHProvable.
generalize (IHProvable F0); intro.
generalize (H0 L0); intro. generalize (H1 w); intro.
simpl. intros.
apply H2.
assert (H6: forall (M: kripke), forall i: agent,
forall x y z: (W (F M)),((R (F M) i x y /\ R (F M) i  y z) -> R (F M) i x z )).
```

```
apply trans.
generalize (H6 {| F := F0; L := L0 |}). simpl. intros.
generalize (H5 i); intro. generalize (H7 w); intro. generalize (H8 y);
intro. generalize (H9 y0); intro. apply H10.  split. exact H3.   exact H4.
```

Правило 22. $\forall n \forall i \in A \forall \phi (n \vdash \neg K_i \phi \Longrightarrow (n \vdash K_i \neg (K_i \phi)))$.

Для произвольной модели $M = (F, L)$, произвольного мира n и агента i верна индуктивная гипотеза:

$IH : M, n \vDash \neg K_i \phi \Longleftrightarrow \neg \forall y (l R_i y \Longrightarrow M, y \vDash \phi)$.

Нам надо доказать, что:

$n \vDash K_i \neg K_i \phi \Longleftrightarrow \forall y (l R_i y \rightarrow \neg \forall z (y R_i z \Longrightarrow M, z \vDash \phi))$.

Вводим следующие гипотезы:

$H : n R_i y$.

$H_1 : M, n \vDash \forall z (y R_i z \Longrightarrow M, z \vDash \phi)$.

Теперь надо доказать, что:

$M, n \vDash \bot$.

Чтобы получить противоречие, требуется доказать:

$M, n \vDash \forall y (l R_i y \Longrightarrow M, y \vDash \phi)$.

Введем гипотезу:

$H_2 : n R_i z$ (y уже занят в H).

Теперь надо доказать, что:

$M, z \vDash \phi$.

Из H_1 получаем, что:

$M, n \vDash (y R_i z \Longrightarrow M, z \vDash \phi)$.

У нас есть аксиомы symm: $\forall i \in A, \forall x, y \in W (x R_i y \rightarrow y R_i x)$ и trans: $\forall i \in A, \forall x, y, z \in W ((x R_i y \& y R_i z) \rightarrow x R_i z)$, верные в моделях нашего типа.

С помощью symm мы получаем из H $y R_i n$, а затем с помощью trans отсюда и из H_2 получаем $y R_i z$. Таким образом, учитывая, что $M, n \vDash (y R_i z \Longrightarrow M, z \vDash \phi)$, получаем $M, z \vDash \phi$.

Правило 23. $\forall n \forall i \in A \forall \phi (n \vdash C \phi \Longrightarrow (n \vdash CC \phi))$.

Корректность данного правила базируется на аксиоме un: $\forall x, y x R_U y$.

Для произвольной модели M и произвольного мира n верна индуктивная гипотеза:

$IH : M, n \vDash C \phi \Longleftrightarrow \forall y (n R_U y \Longrightarrow M, y \vDash \phi)$.

Нам нужно доказать:

$M, n \vDash CC \phi \Longleftrightarrow \forall y (n R_U y \Longrightarrow (\forall z (y R_U z \Longrightarrow M, z \vDash \phi)))$.

Упрощая, получаем, что нам нужно доказать, что $M, z \vDash \phi$.

Применяя индуктивную гипотезу ("apply IH"в Coq), получаем, что нам нужно доказать $nR_U y$.

Для этого применяем аксиому un $\forall x, yxR_U y$, характеризующую универсальное отношение достижимости, откуда сразу же следует требуемое.

Теорема корректности доказана.

Программная реализация теоремы корректности приведена в Приложении 1.

$\lhd$.

Перейдем теперь к формулировке и анализу парадокса "грязных детей".

5 Описание и формализация парадокса "грязных детей"

Сначала неформально опишем указанный парадокс.

Большая группа детей играет в саду (их адекватность, разумность и правдивость подразумевается и поэтому специально не оговаривается). У некоторого числа детей (для определенности, у k) оказываются испачканными грязью лица. Каждый ребенок может видеть грязь на лице других детей, но не на своем собственном. Их отец объявляет, что по крайней мере один ребенок является грязным. Затем он спрашивает их: "Знает ли кто-нибудь из вас, что он является грязным?"Затем отец поторяет вопрос. Дети отвечают "нет"ровно $k-1$ раз, но на k-той итерации вопроса в точности все грязные дети отвечают "да". Чтобы разобраться в данной загадке, разберем несколько случаев для различных значений k.

1. $k = 1$, т.е. у нас только один грязный ребенок. Ребенок немедленно отвечает «да», как только он слышит вопрос отца и видит, что все остальные дети являются чистыми.

2. $k = 2$, скажем, дети a и b являются грязными. Тогда оба отвечают «нет», когда слышат вопрос впервые. Затем a думает: т.к. b в первый раз ответил «нет», то он должен был видеть кого-то еще с грязью на лбу; хорошо, единственный грязный человек, кого я вижу, – это b, так что b имеет в виду именно меня. Так что на вторую итерацию вопроса a отве-

чает "да". b рассуждает симметрично относительно a и также отвечает "да".

3. $k = 3$. Пусть дети a, b и c являются грязными. Каждый из них отвечает "нет"первые два раза. Затем a рассуждает так: если бы только b и c были бы грязными, они ответили бы "да"во второй раз, используя схему, приведенную для случая $k = 2$; так что должен существовать третий грязный ребенок. Т.к. я вижу грязными только b и c, то третьим должен быть я. Так что на третьей итерации вопроса a отвечает "да". Рассуждая симметрично, то же самое делают b и c.

Для остальных значений k дело обстоит аналогичным образом. Заметим, что у нас нет общего знания вплоть до того момента, пока отец не объявит, что кто-то из детей испачкался. Например, в случае $k = 2$ у нас нет общего знания до слов отца, т.к. хотя a и b знают, что кто-то является грязным (они видят друг друга), но, скажем, a не знает, знает ли b, что кто-то еще, кроме a, является грязным, и т.п.

Приведем теперь строгую формализацию указанного парадокса с помощью системы $S5^n$.

Будем использовать обозначение $\bigwedge\limits_{P(a)}{}^{\phi(a)}$ для конъюнкции всех формул вида $\phi(a)$, для которых выполнено условие $P(a)$ (a – некоторый параметр, от которого зависят указанная формула и условие).

Аналогично вводится соответствующее обозначение для дизъюнкции $\bigvee\limits_{P(a)}^{\phi(a)}$.

Введем теперь следующие важные аксиомы, которые позволят оперировать с "большими"дизъюнкцией и конъюнкцией. Пусть S – произвольное множество. $P(a)$ – произвольный унарный предикат, $\phi(a)$ – произвольная формула первого порядка, зависящая от одной свободной переменной. Тогда:

$\forall l \forall S \forall P(a) \forall \phi(a)((\forall s \in S(P(s) \to l \vdash \phi(s))) \Longrightarrow (l \vdash \bigwedge\limits_{P(a)} \phi(a)))$ – введение "большой"конъюнкции $\bigwedge_i$.

$\forall l \forall S \forall P(a) \forall \phi(a)((\exists s \in S(P(s) \& l \vdash \phi(s))) \Longrightarrow (l \vdash \bigvee\limits_{P(a)} \phi(a)))$ – введение "большой"дизъюнкции $\bigvee_i$.

$\forall l \forall S \forall P(a) \forall \phi(a)((l \vdash \bigwedge\limits_{P(a)} \phi(a)) \Longrightarrow (\forall s \in S(P(s) \to l \vdash \phi(s))))$ – удаление "большой"конъюнкции $\bigwedge_e$.

$\forall l \forall S \forall P(a) \forall \phi(a)((l \vdash \bigvee\limits_{P(a)} \phi(a)) \Longrightarrow \exists s \in S(P(s) \& l \vdash \phi(s))))$ – удаление "большой"дизъюнкции $\bigvee_e$.

Смысл этих аксиом-правил очень прост: 1) для конъюнкции – ес-

ли при каждом значении параметра верны и условие, и сама формула $\phi(a)$, то верна "большая условная"конъюнкция формул вида $\phi(a)$, и наоборот; 2) для дизъюнкции – если существует значение параметра, при котором верны и условие, и сама формула $\phi(a)$, то верна и "большая условная"дизъюнкция формул вида $\phi(a)$, и наоборот.

Можно сказать, что это определения конъюнкции и дизъюнкции для схем формул.

Данные правила не верифицируем, т.к. они постулируются как аксиомы, а по сути, они представляют собой просто *определения* для ограниченных кванторов всеобщности и существования (или больших параметрических конъюнкции и дизъюнкции), записанные в виде правил, для простоты применения.

Введем следующее определение:

Определение 18 $\alpha_G := \bigwedge\limits_{i \in G} p_i \wedge \bigwedge\limits_{i \in A \setminus G} \neg p_i$ *при условии, что* $G \subseteq A$.

Данная запись означает, что множество грязных детей есть в точности множество G .

Также будем использовать следующие обозначения:

Определение 19 $\neg\alpha_G$ – *множество грязных детей отлично от* G.

Определение 20 $\bigwedge\limits_{|G|<k} \neg\alpha_G$ – *множество грязных детей имеет мощность не меньше* k.

Для корректной формализации парадокса нам понадбятся следующие леммы (они все доказаны в системе Coq, см. код).

Сделаем такое замечание: отсюда и далее для удобства и более компактной записи будем писать производные правила вывода в виде дроби, где в числителе находятся утверждения о выводимости посылок для любых l (уровень), $i \in A$, а в знаменателе – соответсвующие утверждения о выводимости заключения. Нотацию $\forall l \forall i \in A \vdash ...$ для удобства опускаем; в доказательствах при работе с модальными правилами там, где это необходимо, будем указывать только уровни (возможные миры), относительно которых идет рассуждение.

Лемма 3 (L3): Пусть у нас имеются предикаты P и P' такие, что для всех a $P'(a)$ влечет $P(a)$ (в теоретико-множественном случае речь идет

просто о включении первого множества во второе). Тогда справедливо следующее правило: $\frac{\bigwedge\limits_{P(a)} \bigwedge \phi(a)}{\bigwedge\limits_{P'(a)} \bigwedge \phi(a)}$.

Доказательство

По условию, у нас верно, что:

1. $\forall s(P'(s) \to P(s))$

Так как у нас верно, что выводится $\bigwedge_{P(a)} \phi(a)$, то

2. $\forall s(P(s) \to \phi(s))$ – $\bigwedge_e$.

Нам надо доказать, что $\forall s(P'(s) \to \phi(s))$

3. фиксируем произвольный s.

4. $H : P'(s)$– вводим гипотезу.

Нам надо доказать, что $\phi(s))$ (зависит от H)

5.$P'(s) \to P(s)$ – 3 и 1

6. $P(s) \to \phi(s)$ – 3 и 2

7. $P(s)$ – $\to_e$: 4 и 5.

8. $\phi(s)$ – $\to_e$: 6 и 7.

$\lhd$

Лемма 4 (L4): Для всех агентов i, некоторого предиката P и формулы, зависящей от параметра $\phi(a)$, верно следующее: $\frac{\bigwedge\limits_{P(a)} C(\phi(a))}{\bigwedge\limits_{P(a)} K_i(\phi(a))}$.

Доказательство

1. $\forall s(P(s) \to C\phi(s))$ – $\bigwedge_e$ по условию

Надо доказать, что $\forall s(P(s) \to K_i(\phi(s)))$

2. фиксируем произвольный s.

3. $H : P(s)$– вводим гипотезу

Надо доказать, что $K_i(\phi(s))$ (зависиот от H)

4.$P(s) \to C(\phi(s))$ – 1 и 2

5. $C(\phi(s))$ – $\to_e$: 3 и 4

6. $K_i(\phi(s))$ – CK:5

$\lhd$

Лемма 5 (L5): Для всех агентов i, некоторого предиката P и формулы $\phi(a)$ верно следующее: $\frac{\bigwedge\limits_{P(a)} K_i(\phi(a))}{\bigwedge\limits_{P(a)} \phi(a)}$.

Доказательство аналогично Лемме 4, только на 6 шаге применяем правило рефлексивности T.

Лемма 6 (L6): Для некоторых формул $\phi(a)$, $\psi(a)$ и предиката P верно следующее: $\frac{\bigwedge\limits_{P(a)} \phi(a), \bigwedge\limits_{P(a)} (\phi(a) \to \psi(a))}{\bigwedge\limits_{P(a)} \psi(a)}$.

Доказательство

1. $\forall s(P(s) \to \phi(s))$ – $\bigwedge_e$ по условию

2. $\forall s(P(s) \to (\phi(s) \to \psi(s)))$ – $\bigwedge_e$ по условию

Надо доказать, что $\forall s(P(s) \to \psi(s))$

3. фиксируем произвольный s.

4. $H : P(s)$ – вводим гипотезу

5.$P(s) \to \phi(s)$ – 3 и 1

6. $P(s) \to (\phi(s) \to \psi(s))$ – 3 и 2

7. $\phi(s)$ – $\to_e$: 4, 5

8. $\phi(s) \to \psi(s)$ – $\to_e$: 4, 6

9. $\psi(s)$ – $\to_e$: 7, 8

$\lhd$

Лемма 7 (L7): Для всех агентов i, некоторого предиката P и формулы $\phi(a)$ верно следующее правило вывода: $\frac{\bigwedge\limits_{P(a)} l, K_i(\phi(a))}{K_i \cdots \bigwedge\limits_{P(a)} Increase(l,K_i), \phi(a)} \ldots$, где выражение $K_i...$ в знаменателе означает переход к уровню $Increase(l, K_i)$ для рассматриваемого R_i.

Напомним, что запись l, $K_i(\phi(a))$ означает $l \vdash K_i\phi(a)$.

Заметим, что $Decrease(Increase(l, K_i)) = l$.

Это правило удаления K_i, сформулированное ранее, но расписанное для "большой"конъюнкции.

Доказательство

0. Фиксируем произвольный l, $l\, R_i\, Increase(l, K_i)$.

1. $\forall s(P(s) \to l,\ K_i(\phi(s)))$ – $\bigwedge_e$ по условию.

Надо доказать, что:

$\forall s(P(s) \to Increase(l, K_i),\ \phi(s))$.

2. Фиксируем произвольный s.

Вводим гипотезу:

3. $H : P(s)$

4.$P(s) \to l,\ K_i\phi(s)$ – 2 и 1.

5. $l, K_i\phi(s)$ – $\to_e$: 4, 3

6. $Increase(l, K_i),\ \phi(s)$ – K_iE: 5, 0.

$\lhd$

Лемма 8 (L8): Для некоторой формулы $\phi(a)$ и предиката P верно следующее: $\frac{\neg \bigvee\limits_{P(a)} \phi(a)}{\bigwedge\limits_{P(a)} \neg\phi(a)}$.

Это правило де Моргана для "большой"конъюнкции.

Доказательство

1. $\neg\exists s(P(s)\&\phi(s))$ – $\bigvee_e$ по условию.

Надо доказать, что:

$\forall s(P(s) \to \neg\phi(s))$.

2. Фиксируем произвольный s.

Вводим гипотезу:

3. $H : P(s)$

Теперь надо доказать, что:

$\neg\phi(s)$ (зависит от H).

Вводим гипотезу:

4. $H1 : \phi(s)$

Теперь надо доказать, что:

$\bot$ (зависит от H, $H1$) .

5. $P(s)\&\phi(s)$ – $\&_i$: 3,4

6. $\exists s(P(s)\&\phi(s))$ – из 5.

Доказано, что $\bot$ из 1 и 6.

$\lhd$

Доказательство в Coq данной леммы выглядит так:

```
Lemma L6: forall S:Set, forall l:level, forall phi:S->formula, forall i:agent,
forall cond:S->Prop,
(Provable l (Not(BigOr (cond) (phi))))->
(Provable  l  (BigAnd (cond) (fun s: S => (Not(phi s))))).
intros.
apply BigAndI.
intros.
apply notI.
intros.
assert (H2: Provable l (phi s)). apply Prov. exact H1.
apply notE with (BigOr cond phi).
cut(exists s:S,(cond s)/\(Provable l (phi s))).
intro.
apply BigOrI.
exists s.
```

```
split.
exact H0.
exact H2.
exists s.
split.
exact H0.
exact H2.
exact H.
Qed.
```

Лемма 9 (L9): Для некоторой формулы $\phi(a)$ и предикатов P и Q верно следующее: $\frac{\overset{\wedge}{P(a)}\phi(a), \overset{\wedge}{Q(a)}\phi(a)}{\overset{\wedge}{P(a)\vee Q(a)}\phi(a)}$.

Доказательство

1. $\forall s(P(s) \to \phi(s))$ – $\bigwedge_e$ по условию

2. $\forall s(Q(s) \to \phi(s))$ – $\bigwedge_e$ по условию

Надо доказать, что $\forall s((P(s) \vee Q(s)) \to \psi(s))$

3. фиксируем произвольный s.

4. $H : P(s) \vee Q(s)$ – вводим гипотезу.

Теперь надо доказать, что:

$\psi(s)$ (зависит от H) .

5.$P(s) \to \phi(s)$ – 3 и 1

6. $Q(s) \to \phi(s)$ – 3 и 2

Применяем правило удаления дизъюнкции к 4.

7. $H1 : P(s)$ – вводим гипотезу.

8. $\phi(s)$ – $\to_e$: 5, 7. Зависит от H1.

9. $H2 : Q(s)$– вводим гипотезу.

10. $\phi(s)$ – $\to_e$: 6, 9. Зависит от H1.

11. $\phi(s)$ – $\vee_e$: 4, 8, 10. Зависит только от H, что и требовалось

$\lhd$.

Лемма 10 (L10): Для всех агентов i, некоторого предиката P и формулы $\phi(a)$ верно следующее правило вывода: $\frac{\overset{\wedge}{P(a)} l, C(\phi(a))}{C...\overset{\wedge}{P(a)} Increase(l,), \phi(a)}...,$ где выражение $C...$ в знаменателе означает переход к достижимому миру $Increase(l, C)$ для R_U.

Напомним, что запись $l, C(\phi(a))$ означает $l \vdash C\phi(a)$.

Заметим, что $Decrease(Increase(l, C)) = l$.

Это правило удаления C, сформулированное ранее, но расписанное для "большой"конъюнкции.

Доказательство полностью аналогично Лемме 5, только вместо K_iE на соответствующем шаге используем C_e.

$\lhd$

Приведем теперь формальное доказательство парадокса "грязных детей".

Напомним, что у нас есть конечное множество детей A. Все дети могут видеть друг друга, так что существует общее знание следующего рода: каждый ребенок знает, какие дети грязные, а какие нет (исключая его самого). Данное утверждение можно формализовать посредством аксиом:

$\bigwedge\limits_{i,j\in A, i\neq j} C(p_i \to K_j p_i)$ $(F1)$,

$\bigwedge\limits_{i,j\in A, i\neq j} C(\neg p_i \to K_j \neg p_i)$ $(F2)$.

Как уже упоминалось выше, мы хотим доказать следующее: если в A есть k грязных детей, и $k-1$ раз все они заявляют, что не знают, грязные ли они, то на k-тый раз все грязные дети смогут заключить, что они грязные. Мы докажем это, предварительно доказав следующие два утверждения:

(1) Если у нас есть k $(k \geq 1)$грязных детей и общее знание о том, что этих детей по крайней мере k, то в итоге все грязные дети понимают, что они грязные, в силу того, что они все могут видеть $k-1$ своих грязных товарищей.

(2) Если существует общее знание о том, что у нас по крайней мере k грязных детей, $k \geq 1$, то после заявления о том, что никто не знает, грязный он или нет, общим знанием станет тот факт, что у нас по крайней мере $k+1$ грязный ребенок, т.к. если бы у нас было ровно k грязных детей, то в силу Утверждения 1, все они смогли бы сказать, что они грязные.

Итак, если у нас k грязных детей и отец заявляет, что есть по крайней мере один грязный ребенок, и $k-1$ раз все дети говорят, что не знают, грязные они или нет, то каждый раз мы будем использовать Утверждение 2 (Лемма 12) до тех пор, пока не наступит момент стабилизации и общим знанием не станет тот факт, что у нас по крайней мере k грязных детей. Тогда мы сможем воспользоваться Утверждением 1 (Лемма 11) и доказать, что все грязные дети знают о том, что они грязные.

Докажем первое утверждение.

Оно может быть формализовано следующим образом.

Лемма 11. Пусть множество грязных детей – это $H \subseteq A$, $|H| \geq 1$. Тогда для всех $m \in H$ верно следующее:
$\bigwedge\limits_{i,j \in A, i \neq j} C(p_i \to K_j p_i)$, $\bigwedge\limits_{i,j \in A, i \neq j} C(\neg p_i \to K_j \neg p_i)$, $C(\bigwedge\limits_{|G|<|H|} \neg \alpha_G)$, $\alpha_H \vdash_{S5^n} K_m p_m$.

При доказательстве данной леммы мы будем использовать символ $\wedge_e$ для обозначения того, что мы получаем один из конъюнктов соответствующей "большой"конъюнктивной формулы.

Зафиксируем произвольный уровень (возможный мир) l.

Для ясности изложения, номера допущений, от которых зависят формулы вывода, будем писать рядом с соответствующими формулами в квадратных скобках.

Напомним, что допущение зависит само от себя. Если допущение используется в качестве посылки произвольного правила вывода, то полученная формула также зависит от данного допущения. При применении подходящего непрямого правила вывода зависимость от допущения, которое вводилось специально для работы с данным правилом вывода, устраняется.

Миры, относительно которых идет рассуждение, будем писать сразу перед формулами.

Доказательство.

Докажем утверждение для произвольного $m \in H$.

1. $l, \alpha_H \equiv \bigwedge\limits_{i \in H} p_i \wedge \bigwedge\limits_{i \notin H} \neg p_i$ – посылка.
2. l, $C(\bigwedge\limits_{|G|<|H|} \neg \alpha_G)$ – посылка.

Теперь мы через несколько шагов хотим применить правило C_i и поэтому переходим к уровню (возможному миру) $Increase(l, C), l \, R_U \, Increase(l, C)$. Заметим, что $Decrease(Increase(l, C)) = l$.

3. $Increase(l, C)$, $\bigwedge\limits_{|G|<|H|} \neg \alpha_G$ – $C_e, 2$
4. $Increase(l, C), \neg \alpha_{H \setminus \{m\}}$ – $\wedge_e, 3$.

Мы можем совершить данный переход, т.к. $|H \setminus \{m\}| < |H|$.

Теперь перешли на уровень-предшественник l по отношению R_U:

5. $l, C(\neg \alpha_{H \setminus \{m\}})$ – C_i.
6. $l, K_m(\neg \alpha_{H \setminus \{m\}})$ – C_i – Лемма 4: 5
7. l, $\bigwedge\limits_{i,j \in A, i \neq j} C(\neg p_i \to K_j \neg p_i)$ – посылка.
8. l, $\bigwedge\limits_{i,j \in A, i \neq m} C(\neg p_i \to K_m \neg p_i)$ – $\wedge_e, 7$ (положили $j = m$).
9. l, $\bigwedge\limits_{i \in A, i \notin H} C(\neg p_i \to K_m \neg p_i)$ – Лемма 3: 8.

Лемма 3 оказывается применимой, т.к. $i \notin H$ влечет $i \neq m$.

10. l, $\bigwedge\limits_{i \in A, i \notin H} K_m(\neg p_i \to K_m \neg p_i)$ – Лемма 4: 9.
11. l, $\bigwedge\limits_{i \in A, i \notin H} (\neg p_i \to K_m \neg p_i)$ – Лемма 5: 10.

12. l, $\bigwedge\limits_{i\in A,i\notin H}\neg p_i$ – $\&_e 2,1$.

13. l, $\bigwedge\limits_{i\in A,i\notin H}K_m\neg p_i$ – Лемма 6: 11, 12.

14. l, $\bigwedge\limits_{i,j\in A,i\neq j}C(p_i\to K_j p_i)$ – посылка.

15. l, $\bigwedge\limits_{i,j\in A,i\neq m}C(p_i\to K_m p_i)$ – $\wedge_e$, 14 (положили $j=m$).

16. l, $\bigwedge\limits_{i\in H\setminus\{m\}}C(p_i\to K_m p_i)$ – Лемма 3: 15.

Лемма 3 оказывается применимой, т.к. $i\in H\setminus\{m\}$ влечет $i\neq m$.

17. l, $\bigwedge\limits_{i\in H\setminus\{m\}}K_m(p_i\to K_m p_i)$ – Лемма 4: 16.

18. l, $\bigwedge\limits_{i\in H\setminus\{m\}}(p_i\to K_m p_i)$ – Лемма 5: 17.

19. l, $\bigwedge\limits_{i\in H}p_i$ – $\&_e 2,1$.

20. l, $\bigwedge\limits_{i\in H\setminus\{m\}}p_i$ – Лемма 3: 19.

Лемма 3 здесь оказывается применимой, т.к. $i\in H\setminus\{m\}$ влечет $i\in H$.

21. l, $\bigwedge\limits_{i\in H\setminus\{m\}}K_m p_i$ – Лемма 6: 20, 18.

Далее применим Лемму 7, и поэтому переходим к уровню $Increase(l,K_i)$, $l\,(R_i)\,Increase(l,K_i)$. Заметим, что $Decrease(Increase(l,K_i))=l$.

22. $Increase(l,K_i)$, $\bigwedge\limits_{i\in A,i\notin H}\neg p_i$ – Лемма 7: 13.

Далее начинаем рассуждение от противного, и поэтому вводим следующее допущение (с помощью правила $notI$):

23. $Increase(l,K_i),\neg p_m\equiv\bigwedge\limits_{i\in A,i=m}\neg p_i$, [23] – допущение. Заметим еще раз, что допущение зависит от самого себя.

24. $Increase(l,K_i)$, $\bigwedge\limits_{i\in A\&(i=m\vee i\notin H)}\neg p_i$, [23] – Лемма 9: 22, 23.

25. $Increase(l,K_i)$, $\bigwedge\limits_{i\in A,i\in H\setminus\{m\}}\neg p_i$, [23] – Лемма 3: 24.

Лемма 3 оказывается применимой, т.к. $i\in H\setminus\{m\}$ влечет $i=m\vee i\notin H$, [23]

26. $Increase(l,K_i)$, $\bigwedge\limits_{i\in H\setminus\{m\}}p_i$, [23] – Лемма 7: 21.

27. $Increase(l,K_i)$, $\bigwedge\limits_{i\in H\setminus\{m\}}\neg p_i\wedge\bigwedge\limits_{i\in H\setminus\{m\}}p_i\equiv\alpha_{H\setminus\{m\}}$, [23] – $\&_i$, 25, 26

28. $Increase(l,K_i),\neg\alpha_{H\ \{m\}}$, [23] – C_i – Km_e, 6. Можем применить правило удаления Km, т.к. находимся в достижимом из l по R_i возможном мире (уровне) $Increase(l,K_i)$.

29. $Increase(l,K_i),\bot$, [23] – $\neg_e$, 27, 28.

Так как мы получили на 29-м шаге вывода противоречие, то теперь мы можем ввести отрицание допущения под номером 23. Все последующие формулы в выводе теперь не зависят от него.

30. $Increase(l,K_i),\neg\neg p_m$ – $\neg_i$, 29

31. $Increase(l,K_i),p_m$ – $\neg\neg_e$, 30

В силу того, что на шаге 31 мы получили формулу p_m, то теперь можем применить правило введения K_m и перейти в мир-предшественник l, $l(R_i)Increase(l,K_i)$.

32. $l,K_m p_m$ – $K_m I$, 31.

Лемма доказана.

Последняя формула вывода не зависит ни от какого допущения и справедлива для всех уровней (возможных миров), а значит, и для пустого уровня ϵ.

Заметим, что в нашем доказательстве нам понадобилось множество $H\backslash\{m\}$, т.к. мы наблюдаем в точности m грязных детей. Таким образом, мы можем определить в Coq аксиому «Минус», которая утверждает, что для всех множеств G и агентов i таких, что $G \neq \emptyset$, $i \in G$ существует множество G' такое, что: $|G| = |G'| + 1$, $i \notin G'$, $G' \subset G$, и все агенты из G, кроме i, принадлежат G'.

Второе утверждение

Второе утверждение может быть формализвано и доказано следующим образом.

Лемма12. $\bigwedge\limits_{i,j\in A, i\neq j} C(p_i \to K_j p_i)$, $\bigwedge\limits_{i,j\in A, i\neq j} C(\neg p_i \to K_j \neg p_i)$, $C(\bigwedge\limits_{|G|<n+1} \neg\alpha_G)$, $C(\bigwedge\limits_{i\in A} \neg K_i p_i \wedge \neg K_i \neg p_i)$ $\vdash_{S5^n} C(\bigwedge\limits_{|G|<n+2} \neg\alpha_G)$.

Доказательство.

Доказательство проведем для произвольного n (мощность множества грязных детей) и произвольного уровня (достижимого мира) l.

1. l, $\bigwedge\limits_{i,j\in A, i\neq j} C(p_i \to K_j p_i)$ – посылка.
2. l, $\bigwedge\limits_{i,j\in A, i\neq j} C(\neg p_i \to K_j \neg p_i)$ – посылка.
3. l, $C(\bigwedge\limits_{|G|<n+1} \neg\alpha_G)$ – посылка.
4. l, $C(\bigwedge\limits_{i\in A} \neg K_i p_i \wedge \neg K_i \neg p_i)$ – посылка.
5. l, $CC(\bigwedge\limits_{|G|<n+1} \neg\alpha_G)$ – применяем правило (аксиому транзитивности) 4 к 3.
6. l, $\bigwedge\limits_{i,j\in A, i\neq j} CC(p_i \to K_j p_i)$ – получена из 1 последовательным применением правил $\wedge_e$, $C4$, $\wedge_i$.
7. l, $\bigwedge\limits_{i,j\in A, i\neq j} CC(\neg p_i \to K_j \neg p_i)$ – получена из 2 последовательным применением правил $\wedge_e$, $C4$, $\wedge_i$.

Далее мы применяем Лемму 10 и поэтому переходим к уровню $Increase(l, C)$, достижимому из l по R_U.

8. $Increase(l, C)$, $\bigwedge\limits_{i\in A} \neg K_i p_i \wedge \neg K_i \neg p_i$ – Лемма 10: 4.
9. $Increase(l, C)$, $\bigwedge\limits_{|G|<n+1} \neg\alpha_G$ – C_e, 3. Можем применить правило C_e, т.к. находимся на необходимом и достаточном уровне достижимости.
10. $Increase(l, C)$, $C(\bigwedge\limits_{|G|<n+1} \neg\alpha_G)$ – C_e: 5. Можем применить правило C_e, т.к. находимся на необходимом и достаточном уровне достижимости.
11. $Increase(l, C)$, $\bigwedge\limits_{i,j\in A, i\neq j} C(p_i \to K_j p_i)$ – получена из 6 последовательным применением правил $\wedge_e$, C_e , $\wedge_i$.

12. $Increase(l, C)$, $\bigwedge\limits_{i,j \in A, i \neq j} C(\neg p_i \to K_j \neg p_i)$ – получена из 7 последовательным применением правил $\wedge_e$, C_e , $\wedge_i$.

Введем теперь дополнительное допущение:

13 $Increase(l, C)$, $\bigvee\limits_{|G|=n+1} \alpha_G$, [13]– допущение

На основании данного допущения мы можем утверждать, что существует множество G', $|G'| = n + 1 \neq \emptyset$, и что множество грязных детей есть в точности множество G'.

14. $Increase(l, C), \alpha_{G'}$, [13]. Из 13, по определению $\alpha_{G'}$, вводим новое допущение.

15. $Increase(l, C)$, $\bigwedge\limits_{i \in G'} K_i p_i$, [13], [15] – допущение.

Мы знаем, что $G' \neq \emptyset$, поэтому берем произвольного агента i' из него.

16. $Increase(l, C), \neg K_{i'} p_{i'} \wedge \neg K_{i'} \neg p_{i'}$, [13], [15] – $\wedge_e$, 8.

17. $Increase(l, C), \neg K_{i'} p_{i'}$, [13], [15] – $\&_e 1$, 16.

Теперь вытащим нужный нам конъюнкт из "большой"конъюнкции п.15

18. $Increase(l, C), K_{i'} p_{i'}$, [13], [15] – $\wedge_e$, 15.

19. $Increase(l, C), \bot$, [13], [15] – $\neg_e$, 17, 18.

Теперь введем отрицание допущения под номером 15.

20. $Increase(l, C)$, $\neg \bigwedge\limits_{i \in G'} K_i p_i$, [13] – $\neg_i$, 19.

Теперь применим Лемму 11 для Утверждения 1 ко всем $K_i p_i$ и соберем конъюнкцию из всех результатов ее применения.

21. $Increase(l, C)$, $\bigwedge\limits_{i \in G'} K_i p_i$, [13] – Лемма 11 : 10, 11, 12,14, $\wedge_i$.

22. $Increase(l, C), \bot$, [13] – $\neg_e$: 20, 21.

Теперь введем отрицание второго допущения с номером 13.

23. $Increase(l, C)$, $\neg \bigvee\limits_{|G|=n+1} \alpha_G$ – $\neg_i$, 22.

24. $Increase(l, C)$, $\bigwedge\limits_{|G|=n+1} \neg \alpha_G$ – Лемма 8: 23.

25. $Increase(l, C)$, $\bigwedge\limits_{(|G|=n+1) \vee (|G|<n+1)} \neg \alpha_G$ – Лемма 9: 9, 24.

26. $Increase(l, C)$, $\bigwedge\limits_{|G|<n+2} \neg \alpha_G$ – Лемма 3: 25.

Лемма 3 оказывается применимой, т.к. $|G| < n + 2$ влечет $(|G| = n + 1) \vee (|G| < n + 1)$.

Теперь вводим оператор C и переходим к уровню(миру) l.

27. l, $C(\bigwedge\limits_{|G|<n+2} \neg \alpha_G)$ – C_i, 26.

Лемма 12 доказана.

Она верна для любого возможного мира l, а значит, и для пустого уровня. Последняя формула вывода не зависит ни от какого допущения.

Заметим, что при доказательстве данной Леммы мы не очень строго придерживались правил натурального вывода: в строках 14 – 21 мы

рассуждали о множестве G' и агенте i из этого множества. Мы вывели их существование из п.13, хотя специального правила вывода для этого случая у нас не было. Тем не менее в Coq данную Лемму можно строго формализовать (см. код).

Доказательство Лемм 11 и 12 в Coq технически очень сложное, поэтому мы придерживались близости к программному коду только в главных моментах.

Таким образом, "парадокс грязных детей"доказан (Леммы 11 и 12) и верифицирован, так как Теорема корректности (а точнее, ее программная реализация) верифицировала все правила вывода, используемые при доказательстве Лемм 11 и 12.

Поэтому формализация данного парадокса (см.[1]) является адекватной и корректной.

Его программная реализация приведена в Приложении 2.

6 Связь с теоремой Кузнецова о полноте

Отметим, что доказательства Лемм 11 и 12 в определенном смысле похожи на доказательство критерия распознавания полноты, который, в частности, используется в теореме Кузнецова о полноте, широко применяемой для доказательства полноты систем многозначных логик; точнее, парадокс соответствует случаю полноты многозначной системы.

Обоснуем наше утвержение. Изложение алгоритма распознавания полноты для систем многозначных логик будет вестись по книге [4]Яблонского "Введение в дскретную математику".

Приведем для начала общую формулировку этой теоремы, а затем адаптируем ее для необходимого нам частного случая – парадокса грязных детей.

Теорема. Существует алгоритм для распознавания полноты для системы $B = \{f_1, ..., f_s\}$, где каждая f_l – некоторая формула (функция) от n переменных, аргументы и значения которой принадлежат конечному множеству $\{0 \dots k\}$.

Доказательство.

Построим по индукции последовательность множеств M_0, M_1,...., M_r,... функций не более чем от двух переменных i и j.

Базис индукции. Пусть $M_0 = \emptyset$.

Индуктивный переход. Пусть уже построены множества M_0, M_1,...., M_r. Покажем, как определяется множество M_{r+1}. Для этого выпишем функции, входящие в M_r, $r \geq 0$:

$M_r = \{h_1(i,j), ..., h_{s_r}(i,j)\}$, $s_r = 0$ при $r = 0$; каждая функция из данного множества зависит не более чем от двух переменных.

Для каждого l $(l = 1, ..., s)$ рассмотрим всевозможные формулы вида $f_i(h_1(i,j), ..., h_n(i,j))$, каждая $h_q(i,j) \in M_r$, $1 \geq q \geq s_r$.

Таким образом, просматривая не более чем $s(s_r + 2)^n$ формул, мы, возможно, получим функции, не вошедшие в M_r. Добавим их к M_r и таким образом получим M_{r+1}.

Очевидно, что $M_0 \subseteq M_1 \subseteq ... \subseteq M_r$....

Из построения ясно, что если $M_{r+1} = M_r$, то $M_r = M_{r+1} = ...$, т.е. цепочка множеств стабилизируется. Пусть r^* – минимальный номер, с которого начинается стабилизация.

Тогда, при $r < r^*$, цепочка множеств $M_0 \subset M_1 \subset ... \subset M_r...$ строго возрастает. Так как мощность каждого множества не больше, чем k^{k^2}, где k – множество значений, которые может принимать формула в рассматриваемой многозначной (k-значной) системе, то $r^* \geq k^{k^2}$. Значит, момент стабилизации может быть обнаружен через конечное число шагов. Рассмотрим множество M_{r^*}. Возможны 2 случая:

1) M_{r^*} содержит все функции от двух и менее переменных, а значит, она содержит и функцию Вебба $max(i,j)+1$, которая позволяет выразить все функции, выразимые в B, и поэтому по известному результату для многозначных логик исходная система B полна.

2) M_{r^*} не содержит всех функций от двух и менее переменных, и поэтому исходная система не полна.

Из данных рассуждений легко извлекается алгоритм: строим множества M_0, M_1,...., M_{r^*} до момена стабилизации, затем рассматриваем множество M_{r^*} и по нему определяем, полна система или нет.

Теорема доказна.

Адаптируем данную теорему для нашего парадокса грязных детей.

Под полнотой системы B, сигнатура которой состоит из пропозициональных связок, эпистемических операторов вида K_i и C, а сама она состоит из соответствующего набора аксиом, описывающего начальные условия для парадокса грязных детей (см. начальные условия Леммы 10), в нашем случае будем понимать наличие неподвижной точки для каждого оператора K_i, т.е. доказуемость формул вида $K_i p_i$ $(K_i i)$ для каждого $i \in A$. Это будет аналогом функции Вебба.

Понятие замыкания суперпозиции функций $[f_i(h_1(i,j),...,h_n(i,j))]$ заменим понятием выводимости всех возможных следствий из данного набора аксиом с помощью Лемм 3 – 10; функциональная замкнутость заменяется дедуктивной закнутостью.

Под переменными i, j будем понимать соответствующие индексы, которые используются при формализации парадокса. Очевидно, они могут принимать не более k значений из конечного множества детей A. Также очевидно, что для формализации парадокса нам требуется только два этих индекса. Т.о, переменные p_i, p_j можно заменить на i и j, принимающие k значений, а формулы вида $K_j p_i$ $(K_j i)$ и $K_j \neg p_i$ $(K_j \neg i)$, рассматривать как функции от двух переменных и т.п. Значения данных формул – множества тех миров, где они истинны (многозначность). Формула принимает выделенное значение 1 $(\top)$ тогда и только тогда, когда она истинна во всех мирах из W.

Оператор неподвижной точки $K_i i$ будет тогда функцией от одной переменной.

Лемма 12 (Утверждение 2) описывает индуктивный переход от множества M_r ко множеству M_{r+1} путем добавления формулы (функции от одной переменной), утверждающей, что появилось общее знание о том, что построена функция определенного вида от одной переменной, принимающая на одно значение больше.

Данное пополнение будет продолжаться до тех пор, пока не наступит стабилизация, т.е. не будет построена функция определенного вида, принимающая все k значений и, соответственно, не появится общее знание об этом.

Лемма 11 (Утверждение 1) , необходимая для эффективного пополнения множеств, также является тестом на полноту стабилизировавшейся цепочки множеств: она показывает, что при данных начальных условиях, и при справедливости Леммы 11, и при достижении момента стабилизации, из множества M_{r^*} выводятся формулы $K_i i$ для каждого $i \in A$. (Получена неподвижная точка для каждого оператора K_i – получено соответствующее общее знание; другими словами, все грязные дети знают, что они грязные).

Таким образом, наша аксиоматическая система B полна.

7 Заключение

В работе проанализизировано и несколько модифицировано натуральное исчисление для эпистемической системы $S5^n$, предложенное P. de Wind (см. [1]); также приведена семантика для $S5^n$, которая подходит для программной реализации системы и доказательства теоремы корректности.

В Coq мной была доказана теорема корректности для натурального исчисления $S5^n$, и с ее помощью верифицировано доказательство эпистемического парадокса "грязных детей".

Показано, что доказательство парадокса аналогично доказательству критерия полноты для многозначных логик (используется в теореме Кузнецова о полноте). Так как похожие алгоритмы применяются в компьютерных науках, данное наблюдение напоминает, что удобные натуральные системы эпистемической логики могут быть полезны для верификации программ.

Список литературы

[1] P. de Wind. Modal logic in Coq. Preprint edition, 2001.

[2] C. Benzmuller, B. W. Paleo. Interacting with modal logics in the Coq proof assistant. Preprint edition, 2014.

[3] D.M. Gabbay, A. Kurucz, F. Wolter and M. Zakharyaschev, Many-Dimensional Modal Logics: Theory and Applications, Elsevier, Studies in Logic and the Foundations of Mathematics, volume 148, 2003, ISBN 0-444-50826-0.

[4] С. В. Яблонский. Введение в дискретную математику. М., 2003.

8 Приложение 1. Система $S5^n$ в Coq: ее формулировка, семантика и натуральное исчисление. Теорема корректности для указанной системы натурального вывода.

```
Require Export List.
Require Import Classical.
Export ListNotations.
Set Implicit Arguments.

Definition agent:=nat.
Inductive var:Set:=p:agent->var.
Inductive formula:Set:=
Top : formula
|Bottom : formula
|Atom : var->formula
|Not : formula->formula
|And : formula->formula->formula
|Or : formula->formula->formula
|Imp : formula->formula->formula
|BiImp : formula->formula->formula
|K : agent->formula->formula
|C : formula->formula.

Infix  "-->"  := Imp (at level 8, right associativity).
Infix "||" := Or.
Infix "&&" := And.

Definition BiImp1 phi psi: formula:=
(phi --> psi) && (psi --> phi).

Infix "<-->" := BiImp1 (at level 8, right associativity).

Record frame:Type:={
W : Set; (* worlds *)
R: agent->(W->(W->Prop));
```

```
RU: W->(W->Prop)}.
Record kripke: Type:={
F : frame; (* F=(W,R, RU) *)
L : (W F)->var->Prop}.


Fixpoint satisfies (M: kripke)(x :(W (F M))) (phi:formula) :Prop:=
match phi with
| Top => True
| Bottom => False
| (Atom phi)  => (L M x phi)
| (Not phi) => ~(satisfies M x  phi)
| (And phi_1 phi_2) =>
(satisfies M x  phi_1) /\ (satisfies M x  phi_2)
|(Or phi_1 phi_2) =>
(satisfies M x  phi_1) \/ (satisfies M x  phi_2)
|(Imp phi_1 phi_2) =>
(satisfies M x  phi_1) -> (satisfies M x  phi_2)
|(BiImp phi_1 phi_2) =>
(satisfies M x  phi_1) <-> (satisfies M x  phi_2)
|(K i phi) =>
(forall y:(W (F M)),(R (F M) i x y) -> (satisfies M y  phi))
|(C  phi) =>
(forall y:(W (F M)), (RU (F M)  x y) -> (satisfies M y  phi))
end.

Definition M_satisfies M phi  := forall w:(W (F M)), (satisfies M w  phi).

Definition valid_in_frame F phi :=
forall L, (M_satisfies (Build_kripke F L) phi).
(* |= phi *)
Definition valid phi:= forall F, (valid_in_frame F phi).
(*Γ|= phi*)
Definition Satisfies F L Γ  w := forall phi, In phi Γ
 -> satisfies (Build_kripke F L) w phi.

Definition Entailment F Γ  L psi :=   forall w, (Satisfies F L Γ  w
 -> satisfies (Build_kripke F L) w psi).
```

```
Definition Entailment1 Γ  phi := forall F,forall L,  Entailment F Γ  L phi.
Definition Entailment2 phi := Entailment1 nil phi.

(*Axioms of reflexivity, transitivity, universal modality and symmetry*)

Axiom reff:  forall (M: kripke), forall i: agent,
forall y: (W (F M)),(R  (F M) i y y).

Axiom trans:  forall (M: kripke), forall i: agent,
 forall x y z: (W (F M)),((R (F M) i x y /\ R (F M) i y z) -> R (F M) i x z ).

Axiom un:  forall (M: kripke),  forall x y : (W (F M)),(RU(F M)  x y ).

Axiom symm:  forall (M: kripke), forall i: agent,
forall x y: (W (F M)),(R (F M) i x y  -> R (F M) i y x).

Theorem reflax:  forall phi: formula,
forall i: agent, valid  ((K i phi) -->  phi).
intros.
unfold valid.
unfold valid_in_frame.
unfold M_satisfies.
intros.
simpl in |-*.
intros.
generalize (H w); intro.
apply H0.
assert (H3: forall (M: kripke),
forall i: agent, forall x : (W (F M)),(R (F M) i x x)).
apply reff.
generalize (H3 (Build_kripke F0 L0)).
intros.
simpl.
generalize (H1 i); intros.
generalize (H2 w); intro.
simpl in  H4.
exact H4.
Qed.
```

```
Theorem pseudotrans:  forall phi: formula, forall i j: agent,
valid ((C phi) -->(K i phi) --> (K j (K i phi))).
intros.
unfold valid.
intros.
unfold valid_in_frame.
intros.
unfold M_satisfies.
intros.
simpl in |-*.
intros.
apply H.
assert (H3: forall (M: kripke),
forall x y : (W (F M)),(RU (F M) x y)).
apply un.
generalize (H3 (Build_kripke F0 L0)).
intros.
simpl.
apply H4.
Qed.

(*Provability*)


Inductive label : Set :=
o : label | k : agent->label | c: label.
(* k and c
are labels for the modal
connectives K and C *)
Inductive level : Set :=
nil : level |
cons : label -> level -> level.
Definition Increase (l:level) (lab: label) :level :=(cons lab l).
(* add a label in front of the list *)
Fixpoint Decrease (l:level): level:=
match l with
|nil => nil|
```

```
(cons o l') => (cons o (Decrease l'))|
(cons (k i) l') => (cons o l')|
(cons c l') => (cons o l')
end.
(* replace the first modal
connective label in the list with o *)
Fixpoint Remove_o (l: level): level :=
match l with
|nil => nil |
(cons e l') => match e with
|o => (Remove_o l') |
(k i) => (cons e l')|
c => (cons e l')
end
end.
(* remove all o's at the front of the list *)


Fixpoint Check_c (l:level): Prop:=
match l with
nil => False |
(cons o l') => (Check_c l') |
(cons (k i) l') => False |
(cons c l') => True
end.
Fixpoint Check_k(i:agent) (l:level): Prop:=
match l with
nil => False |
(cons o l') => (Check_k i l') |
(cons c l') => False |
(cons (k a) l') => i=a
end.
Definition EqLevel (l l': level) :=(Remove_o l)=(Remove_o l').
(*two lists are equivalent if they are the same after
removing all o's at the front of the lists *)
Parameter Ass:level->formula->Prop.

Inductive Provable: level->formula->Prop:=
```

```
|truth : forall n:level,(Provable n Top)
|ass_at :  forall phi:formula, forall l m:level,
(EqLevel l m) -> ((Provable l phi) -> (Provable  m phi))
|andI :  forall phi1 psi2:formula, forall n:level,
(Provable n phi1) -> (Provable  n psi2) ->
(Provable  n (And phi1 psi2))
|andE1 :  forall phi1 phi2:formula, forall n:level,
(Provable n (And phi1 phi2)) -> (Provable n phi1)
|andE2 : forall phi1 phi2:formula, forall n:level,
(Provable  n (And phi1 phi2)) -> (Provable  n phi2)
|orI1 :  forall phi1 phi2:formula, forall n:level,
(Provable  n phi1) -> (Provable n (Or phi1 phi2))
|orI2 :  forall phi1 phi2:formula, forall n:level,
(Provable n phi2) -> (Provable n (Or phi1 phi2))
|orE :  forall phi1 phi2 psi:formula, forall n:level,
(Provable  n (Or phi1 phi2)) ->
(Provable  n (phi1 --> psi)) ->
(Provable n  (phi2 --> psi)) -> (Provable  n psi)
|impI : forall phi1 phi2:formula, forall n:level,
((Ass n phi1) -> (Provable n phi2))->
(Provable  n (Imp phi1 phi2))
|impE :  forall phi1 phi2:formula,forall n:level,
(Provable  n (Imp phi1 phi2)) ->
(Provable  n phi1) -> (Provable  n phi2)
|notI : forall phi:formula, forall n:level,
(Provable n (phi --> Bottom)) ->
(Provable  n (Not phi))
|notE :  forall psi:formula, forall n:level,
(Provable  n psi) ->
((Provable  n (Not psi)) -> (Provable  n Bottom))
|botE : forall phi:formula, forall n:level,
(Provable  n Bottom) -> (Provable  n phi)
|notnotE :  forall phi:formula, forall n:level,
(Provable  n (Not (Not phi))) -> (Provable  n phi)
|KiE :  forall phi:formula, forall i: agent, forall n:level,
((Check_k i n) -> (Provable (Decrease n) (K i phi)) -> (Provable n phi))
|KiI :  forall phi:formula, forall i: agent, forall n:level,
(Provable  (Increase n(k i)) phi) -> (Provable  n (K i phi))
```

```
|CK:  forall phi:formula, forall i: agent, forall n:level,
(Provable  n (C phi)) -> (Provable  n (K i phi))
|CE :  forall phi:formula, forall i: agent, forall n:level,
((Check_c n) -> (Provable  (Decrease n) (C phi)) -> (Provable  n phi))
|CI :  forall phi:formula, forall i: agent,forall n:level,
(Provable (Increase n c) phi) -> (Provable  n (C phi))
|t :  forall phi:formula, forall i: agent, forall n:level,
(Provable  n (K i phi)) -> (Provable n phi)
|four :  forall phi:formula, forall i: agent, forall n:level,
(Provable  n (K i phi)) -> (Provable  n (K i (K i phi)))
|five :  forall phi:formula, forall n:level, forall i: agent,
(Provable  n (Not (K i phi))) ->
(Provable  n (K i (Not (K i  phi))))
|Cf :  forall phi:formula, forall i: agent, forall n:level,
(Provable  n (C phi)) -> (Provable  n (C (C phi))).

Lemma mod2:  forall phi psi: formula, forall i: agent,
 Provable  nil ((K i((phi --> psi))) --> ((K i phi) -->  (K i psi))).
intros. apply impI. intro.
apply impI. intro.
apply KiI. unfold Increase.
assert (H1: Provable nil (K i phi)).
apply Prov. exact H0.
assert (H2: Provable nil (K i (phi --> psi))).
apply Prov. exact H.
assert (H3: Provable (cons (k i) nil )(phi --> psi)).
apply KiE with i. unfold Check_k.
tauto. unfold Decrease. simpl in |-*.
apply ass_at with nil.
unfold EqLevel.
unfold Remove_o.
reflexivity.
auto.
assert (H4: Provable (cons (k i) nil) phi).
apply KiE with i. unfold Check_k.
tauto. unfold Decrease. simpl in |-*.
apply ass_at with nil.
unfold EqLevel.
```

```
unfold Remove_o.
reflexivity.
auto.
apply impE with phi.
auto. auto.
Qed.

Lemma mod5:  forall phi psi: formula, forall i: agent,
Provable  nil (phi --> phi) --> (K i (phi --> phi)).
intros. apply impI. intro.
apply KiI.
apply impI.
intros.
apply Prov.
auto.
Qed.

(*A postulate to operate with satisfiable assumptions*)

Axiom IAss: forall phi psi:formula, forall F, forall L,
forall l: level, forall w, (((Ass l phi) ->
satisfies (Build_kripke F L) w psi) /\ satisfies (Build_kripke F L) w phi)
->   satisfies (Build_kripke F L) w psi.

(*This technical  axiom states that two arbitrary levels differing only in
 leading 0's are equal*)

Axiom Check: forall n: level, cons o n = n.

Definition Prop_Soundness :=  forall phi:formula,
Provable  nil  phi -> valid phi.

Theorem Soundness: Prop_Soundness.
unfold Prop_Soundness. intros.
induction H.  simpl;intros.

(*True*)
```

```
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. unfold satisfies.
intuition.

(*Assumption at equal levels*)
auto.

(*Conjunction1*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.  split.
unfold valid in IHProvable1.
unfold valid_in_frame in IHProvable1.
unfold M_satisfies in IHProvable1.  apply IHProvable1.
unfold valid in IHProvable2.
unfold valid_in_frame in IHProvable2.
unfold M_satisfies in IHProvable2.  apply IHProvable2.

(*Conjunction2*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.  apply IHProvable.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.  apply IHProvable.

(*Disjunction1*)

unfold valid. intros. unfold valid_in_frame. intros.
unfold M_satisfies. intros. left.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.  apply IHProvable.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros. right.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
```

```
unfold M_satisfies in IHProvable.  apply IHProvable.

(*Disjunction2*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable1.
unfold valid_in_frame in IHProvable1.
unfold M_satisfies in IHProvable1.
unfold valid in IHProvable2.
unfold valid_in_frame in IHProvable2.
unfold M_satisfies in IHProvable2.
unfold valid in IHProvable3.
unfold valid_in_frame in IHProvable3.
unfold M_satisfies in IHProvable3.
generalize (IHProvable1 F0); intro.
generalize (H2 L0); intro.
generalize (H3 w); intro.
generalize (IHProvable2 F0); intro.
generalize (H5 L0); intro.
generalize (H6 w); intro.
generalize (IHProvable3 F0);
intro. generalize (H8 L0);
intro. generalize (H9 w); intro.
simpl in H10.
simpl in H7.
elim H4.
auto.
auto.

(*Implication1*)

unfold valid. intros. unfold valid_in_frame. intros.
unfold M_satisfies. intros.
try simpl in |-*. intro.
unfold valid in H0.  unfold valid_in_frame in H0.
unfold M_satisfies in H0.
apply IAss with  phi1 n.
```

```
split.
intro.
apply H0.
auto.
auto.


(*Implication2*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable1.  unfold valid_in_frame in IHProvable1.
unfold M_satisfies in IHProvable1.
unfold valid in IHProvable2.  unfold valid_in_frame in IHProvable2.
unfold M_satisfies in IHProvable2.
simpl in IHProvable1.
generalize (IHProvable2 F0); intro.
generalize (H1 L0); intro. generalize (H2 w); intro.
generalize (IHProvable1 F0); intro.
generalize (H4 L0); intro. generalize (H5 w); intro.
apply H6.
auto.

(*Negation1*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
try simpl in |-*. intro.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
simpl in IHProvable.
apply IHProvable with F0 L0 w.
auto.

(*Bottom1*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
```

```
unfold valid in IHProvable1.  unfold valid_in_frame in IHProvable1.
unfold M_satisfies in IHProvable1.
unfold valid in IHProvable2.  unfold valid_in_frame in IHProvable2.
unfold M_satisfies in IHProvable2.
simpl in IHProvable2. simpl in |-*.
generalize (IHProvable1 F0); intro. generalize (H1 L0);
intro. generalize (H2 w); intro.
generalize (IHProvable2 F0); intro. generalize (H4 L0);
intro. generalize (H5 w); intro.
apply H6.
auto.

(*Bottom2*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
generalize (IHProvable F0); intro. generalize (H0 L0);
intro. generalize (H1 w); intro.
contradict H2.



(*Double Negation*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl in IHProvable.
generalize (IHProvable F0); intro.
generalize (H0 L0); intro. generalize (H1 w); intro.
assert (H4:  ~ ~ satisfies {| F := F0; L := L0 |} w phi
-> satisfies {| F := F0; L := L0 |} w phi).
intro.
assert ((satisfies {| F := F0; L := L0 |} w phi)
\/~ satisfies {| F := F0; L := L0 |} w phi) as H6.
```

```
apply classic.
elim H6; intro.
assumption.
contradict H3.
auto.
apply H4.
auto.

(*Ki1*)

induction n.
simpl in H0.
simpl in H.
contradict H.
induction l.
simpl in H.
try simpl in H0.
apply IHn.
auto.
apply ass_at with (cons o (Decrease n)).
unfold Decrease.
unfold EqLevel.
unfold Remove_o.
reflexivity.
exact H0.
simpl in H.
simpl in IHn.
simpl in H0.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl in IHProvable.
generalize (IHProvable F0); intro. generalize (H1 L0);
intro. generalize (H2 w); intro.
apply H3.
assert(H5: forall (M: kripke), forall i: agent,
forall x: (W (F M)), (R (F M)i x x)).
```

```
apply reff.
generalize (H5 {| F := F0; L := L0 |}). simpl. intros.
apply H4.
simpl in H.
contradict H.

(*Ki2*)

induction n.
simpl in H.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.
unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl in IHProvable.
intros.
generalize (IHProvable F0); intro. generalize (H0 L0); intro.
simpl. intros.
apply H1.
induction l.
simpl in H.
simpl in IHn.
apply IHn.
apply ass_at with (Increase (cons o n) (k i)).
unfold EqLevel.
unfold Remove_o.
simpl.
subst.
assert (H0: cons o n = n).
apply Check.
symmetry in H0.
rewrite <- H0.
tauto.
exact H.
simpl in H.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
```

```
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl in IHProvable.
intros.
generalize (IHProvable F0); intro. generalize (H0 L0); intro.
simpl. intros.
apply H1.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl in IHProvable.
intros.
generalize (IHProvable F0); intro. generalize (H0 L0); intro.
simpl. intros.
apply H1.

(*CK*)

unfold valid. intros. unfold valid_in_frame. intros.
unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl.
try simpl in IHProvable.
generalize (IHProvable F0); intro.
generalize (H0 L0); intros. generalize (H1 w); intro.
apply H3.
assert (H5:forall (M: kripke),  forall x y : (W (F M)),(RU(F M)  x y )).
apply un.
generalize (H5 {| F := F0; L := L0 |}). simpl. intros.
apply H4.

(*C1*)

induction n.
simpl in H0.
simpl in H.
```

```
contradict H.
induction l.
simpl in H.
try simpl in H0.
apply IHn.
auto.
apply ass_at with (cons o (Decrease n)).
unfold Decrease.
unfold EqLevel.
unfold Remove_o.
reflexivity.
exact H0.
induction i.
simpl in H.
contradict H.
auto.
simpl in IHn.
simpl in H0.
subst.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl.
try simpl in IHProvable.
assert (H5:forall (M: kripke),
forall x y : (W (F M)),(RU(F M)  x y )).
apply un.
generalize (IHProvable F0); intro.
generalize (H1 L0); intros. generalize (H2 w); intro.
apply H3.
generalize (H5 {| F := F0; L := L0 |}). simpl. intros.
apply H4.

(*C2*)

induction n.
simpl in H.
```

```
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl in IHProvable.
intros.
generalize (IHProvable F0); intro. generalize (H0 L0); intro.
simpl. intros.
apply H1.
induction l.
simpl in H.
simpl in IHn.
apply IHn.
apply ass_at with (Increase (cons o n) (c)).
unfold EqLevel.
unfold Remove_o.
simpl.
subst.
assert (H0: cons o n = n).
apply Check.
symmetry in H0.
rewrite <- H0.
tauto.
exact H.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl.
intros.
generalize (IHProvable F0); intro. generalize (H1 L0); intro.
generalize (H2 y); intro.
auto.
unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
try simpl.
```

```
intros.
generalize (IHProvable F0); intro. generalize (H1 L0); intro.
generalize (H2 y); intro.
auto.

(*reflexivity*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
simpl in IHProvable.
generalize (IHProvable F0); intro.
generalize (H0 L0); intro. generalize (H1 w); intro.
apply H2.
assert (H5: forall (M : kripke)(i: agent)
(y : W (F M)), R (F M) i y y).
apply reff.
generalize (H5 {| F := F0; L := L0 |}). simpl. intros.
generalize (H3 i). intro. apply H4.

(*transitivity*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.
unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
simpl in IHProvable.
generalize (IHProvable F0); intro.
generalize (H0 L0); intro. generalize (H1 w); intro.
simpl. intros.
apply H2.
assert (H6: forall (M: kripke), forall i: agent,
forall x y z: (W (F M)),((R (F M) i x y /\ R (F M) i  y z)
-> R (F M) i x z )).
apply trans.
generalize (H6 {| F := F0; L := L0 |}). simpl.
```

```
intros. generalize (H5 i); intro. generalize (H7 w); intro.
generalize (H8 y); intro. generalize (H9 y0); intro.
apply H10.  split. exact H3.   exact H4.

(*symmetry*)

unfold valid. intros. unfold valid_in_frame.
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
simpl in IHProvable.
generalize (IHProvable F0); intro. generalize (H0 L0); intro.
generalize (H1 w); intro.
simpl. intros.
intro.
apply H2.
intro.
assert (H7: forall (M: kripke), forall i:agent ,
forall x y z: (W (F M)),((R (F M) i x y /\ R (F M) i y z)
-> R (F M) i x z)).
apply trans.
generalize (H7 {| F := F0; L := L0 |}). simpl. intros.
generalize (H5 i); intro. generalize (H8 y).
 intro. generalize (H9 w);intro.
generalize (H10 y0);intro.
apply H4.
apply H11. split.
assert (H13: forall (M: kripke), forall i:agent ,
forall x y: (W (F M)),(R (F M) i x y  -> R (F M) i y x)).
apply symm.
generalize (H13 {| F := F0; L := L0 |}). simpl. intros.
generalize (H12 i); intro. generalize (H14 w). intro.
generalize (H15 y);intro.  apply H16. auto.
auto.

(*transitivity for C*)

unfold valid. intros. unfold valid_in_frame.
```

```
intros. unfold M_satisfies. intros.
unfold valid in IHProvable.  unfold valid_in_frame in IHProvable.
unfold M_satisfies in IHProvable.
simpl in IHProvable.
generalize (IHProvable F0); intro. generalize (H0 L0); intro.
generalize (H1 w); intro.
simpl. intros.
apply H2.
assert (H6:forall (M: kripke),  forall x y : (W (F M)),(RU(F M)  x y )).
apply un.
generalize (H6 {| F := F0; L := L0 |}). simpl.
intros. generalize (H5 w); intro.
generalize (H7 y0); intro. exact H8.

Qed.
```

9 Приложение 2. Доказательство "парадокса грязных детей"в Coq.

```
Load "C:\Program Files (x86)\Coq\bin\KripkeP33". (* load the Coq
code*)
(* We define a type set to represent sets of agents
and a few functions on sets. *)
Require Export List.
Require Import Classical.
Export ListNotations.
Set Implicit Arguments.


Inductive set : Set := nil' : set |
cons' : agent -> set -> set.
Fixpoint In (l:set) (i : agent): Prop := (* l  Includes i *)
match  l with
|nil' => False
| (cons' j m) => (j= i)\/(In m i)
end.
Fixpoint Size (l:set) : nat:=
match l with
|nil' => 0
|(cons' e l') => (S (Size l'))
end.
Definition Incl (l m: set) := forall i:agent, ((In m i)->(In l i)).
(* l Includes m *)

(* (BigOr S P phi) is the disjunction of all formulas (phi s)
with s:S and (P s) *)
Parameter BigOr: forall S: Set,
(S->Prop)->(S->formula)->formula.
(*
The following axioms say that if an s exists
 with property P we
have that (phi s) is provable, then we have that
```

```
(BigOr S P phi) is provable, and vice versa
*)
Axiom BigOrI:  forall S: Set, forall P: S ->Prop,
forall phi:S ->formula, forall n:level,
(exists s: S, (P s)/\(Provable  n (phi s))) ->
(Provable  n (BigOr (P) (phi))).

Axiom BigOrE:  forall S: Set, forall P: S ->Prop,
forall phi:S ->formula, forall n:level,
(Provable  n (BigOr(P) (phi) ))
->(exists s: S, (P s)/\(Provable n (phi s))).

(*BigAnd S P phi) is the
conjunction of all formulas (phi s)
with s:S and (P s) *)
Parameter BigAnd:forall S: Set,
(S -> Prop)->(S -> formula)->formula.
(*
The following axioms say that if for all s with property P we
have that (phi s) is provable, then we have that
(BigAnd S P phi) is provable, and vice versa
*)
Axiom BigAndI: forall S: Set, forall P: S ->Prop,
forall phi:S ->formula, forall n:level,
(forall s: S, (P s)->(Provable  n (phi s))) ->
(Provable  n (BigAnd(P) (phi) )).

Axiom BigAndE: forall S: Set, forall P: S ->Prop,
forall phi:S ->formula, forall n:level,
(Provable  n (BigAnd(P) (phi) ))->(forall s: S, (P s)->(Provable  n (phi s))).

Definition muddy (i:agent): formula := (Atom (p i)).
Definition not_muddy (i: agent): formula:= (Not (muddy i)).
Definition uneq (A: Set) := forall x y: A,(eq x y)->False.
Definition NotIn (l: set):= forall x: agent, (In l x)->False.

Check (BigAnd (In
(cons' (S O) (
```

```
cons' (S(S O)) (
cons' (S(S(S O))) nil'))))
(fun x: agent =>(muddy O)-->(K x (muddy O)))).

(*We will prove two entailments that
can be used to prove
that if there are n muddy
children, and n-1 times everyone
announces that they don't know whether they are muddy, all the
muddy children can
conclude they are muddy*)

Require Classical. (*
classic
: (P:Prop)P\/~P *)
(* We use the following 8 lemmas to reason with the
BigAnd and BigOr: *)

Lemma L1: forall S:Set, forall l:level, forall phi:S->formula,
forall cond cond':S->Prop, (Provable  l (BigAnd (cond) (phi))) ->
(forall j:S,(cond' j)->(cond j)) ->
(Provable l (BigAnd (cond') (phi))).
intros.
apply BigAndI.
intros.
apply BigAndE with cond.
exact H.
apply H0; assumption.
Qed.

Lemma L2: forall S:Set,forall  l:level,
forall phi:S->formula, forall i:agent,
forall cond:S->Prop,
(Provable  l (BigAnd (cond) (fun j: S => (C (phi j)))))->
(Provable  l (BigAnd (cond)(fun j: S => (K i (phi j))))).
intros.
apply BigAndI.
intros.
```

```
apply CK.
change (Provable  l ((fun j : S => (C (phi j)))s)).
apply BigAndE with cond.
exact H.
exact H0.
Qed.

Lemma L3: forall S:Set, forall l:level,
forall phi:S->formula, forall i:agent,
forall cond:S->Prop,
(Provable  l (BigAnd (cond) (fun j: S => (K i (phi j)))))->
(Provable  l (BigAnd (cond) (phi))).
intros.
apply BigAndI.
intros.
apply t with i.
change (Provable  l ((fun j : S => (K i (phi j)))s)).
apply BigAndE with cond.
exact H.
exact H0.
Qed.


Lemma L4: forall S:Set, forall l:level,
forall phi psi:S->formula, forall i:agent,
forall cond:S->Prop,
(Provable  l (BigAnd (cond) (phi)))->
(Provable l (BigAnd (cond)(fun j: S => ((phi j) --> (psi j)))))
->(Provable  l (BigAnd (cond) (psi))).
intros.
apply BigAndI.
intros.
apply impE with (phi s).
change (Provable  l ((fun j : S => ((phi j) --> (psi j)))s)).
apply BigAndE with cond.
exact H0.
exact H1.
apply BigAndE with cond.
```

```
exact H.
exact H1.
Qed.

Lemma L5: forall S:Set, forall l:level, forall phi:S->formula, forall i:agent,
forall cond:S->Prop,
(Provable  (Increase (l)(o)) (BigAnd (cond) (fun j: S => (K i (phi j)))))->
(Provable  (Increase (l)(k i)) (BigAnd (cond) (phi))).
intros.
apply BigAndI.
intros.
apply KiE with i.
unfold Check_k.
reflexivity.
simpl in |-*.
change (Provable  (Increase l o) ((fun j : S => (K i (phi j)))s)).
apply BigAndE with cond.
exact H.
exact H0.
Qed.

Lemma L6: forall S:Set, forall l:level, forall phi:S->formula, forall i:agent,
forall cond:S->Prop,
(Provable l (Not(BigOr (cond) (phi))))->
(Provable  l  (BigAnd (cond) (fun s: S => (Not(phi s))))).
intros.
apply BigAndI.
intros.
apply notI.
simpl in |-*.
apply impI.
intros.
assert (H2: Provable l (phi s)). apply Prov. exact H1.
apply notE with (BigOr cond phi).
cut(exists s:S,(cond s)/\(Provable l (phi s))).
intro.
apply BigOrI.
exists s.
```

```
split.
exact H0.
exact H2.
exists s.
split.
exact H0.
exact H2.
exact H.
Qed.

Lemma L7: forall S:Set, forall l:level, forall phi:S->formula, forall i:agent,
forall cond cond':S->Prop,
(Provable  l (BigAnd (fun g:S => (cond g)) (phi)))->
(Provable  l (BigAnd (fun g:S => (cond' g)) (phi)))->
(Provable  l (BigAnd (fun g:S => ((cond g)\/(cond' g))) (phi))).
intros.
apply BigAndI.
intros.
elim H1.
intro.
apply BigAndE with (fun g:S => (cond g)).
exact H.
auto.
intro.
apply BigAndE with (fun g:S => (cond' g)).
exact H0.
auto.
Qed.


Lemma L8: forall S:Set, forall l:level, forall phi:S->formula, forall i:agent,
forall cond:S->Prop,
(Provable (Increase (l)(o)) (BigAnd (cond) (fun j: S => (C (phi j)))))->
(Provable  (Increase (l)(c)) (BigAnd (cond) (phi))).
intros.
apply BigAndI.
intros.
apply CE.
```

```
auto.
unfold Check_c.
reflexivity.
simpl in |-*.
change (Provable  (Increase l o) ((fun j : S => (C (phi j)))s)).
apply BigAndE with cond.
exact H.
exact H0.
Qed.

Parameter A:set. (* Suppose the group of
children is A *)
Definition alpha (G: set):formula:=
(BigAnd (fun i: agent => ((In A i)/\(In G i))) (muddy)) &&
(BigAnd (fun i:agent =>((In A i)/\((In G i)->False)))(not_muddy)).
(* (alpha H) means exactly the
children In H are muddy *)

Section Entailment_1.
(*
In this section we prove the first entailment: if there are n
children with mud on their foreheads, and it is
common knowledge
that there are at least n
children with mud on their foreheads,
then they all know they are muddy, since they
can see each other
*)

Variable L:level.
(* Our "start level" for the proof is L (see S5n) *)


Variable m:agent.

Hypothesis In_A_m: forall m: agent, (In A m).
(* Take an arbitrary agent m *)
(* from the group A of
```

```
children *)

Variable  H: set. (* Take a set H of agents *)

Hypothesis Incl_A_H: forall H: set, (Incl A H).
(* such that H is a subset of A *)
Hypothesis In_H_m:  forall H: set, (In H m).
(* and m is an element of H *)

(* and exactly the
children in H are muddy *)
(* Suppose it is
common knowledge that the number
of muddy
children is at least the size of H *)
Definition Size_lt(n:nat) (G:set):Prop:=(lt (Size G) n).

Axiom setminus:
forall G:set, forall i:agent, (* For all sets G and agents i such that *)
(In G i) -> (* G includes i, *)
(exists G':set, (* a set G' exists, such that *)
(S (Size G'))=(Size G) (* the size of G = the size of G'+1, *)
/\ ~(In G' i) (* and i is not an element of G', *)
/\ (Incl G G')(* and G' is a subset of G, *)
/\ forall j:agent,(In G j)->(j<> i<->(In G' i))).
(* and all agents from G besides i are in G' *)
(* so G'=G\{i} *)

Lemma entailment_1: forall l: level,(Provable  (Increase (l) (o))
(BigAnd (In A)
(fun i:agent => (BigAnd (fun j:agent=> (In A j)/\(i<>j))
(fun j:agent=>(C (muddy j)-->(K i (muddy j))))))))
->(Provable   (Increase (l) (o))
(BigAnd (In A)
(fun i:agent =>
 (BigAnd (fun j:agent=> (In A j)/\(i<>j))
(fun j:agent=>(C (not_muddy j)
-->(K i (not_muddy j))))))))->
```

```
((fun H: set=>
(Provable (Increase (l) (o)) (alpha H)))H)->
((fun H: set=>(Provable (Increase(Increase (l) (o)) (o))
(C (BigAnd  (fun G: set =>(Incl A G)/\(Size_lt (Size H) G))
(fun G:set=>(Not(alpha G)))))))H)->
Provable l (K m (muddy m)).
intro.
intro see_muddy_others.
intro see_clean_others.
intro muddy_H.
intro at_least_size_H.
apply KiI.
apply notnotE.
apply notI.
apply impI; intro.
assert (H1: Provable (Increase l (k m)) (Not (muddy m))).
apply Prov. exact H0.
clear H0.
elim (setminus H m).
intro H'.
intros.
elim H0.
intros.
elim H3.
intros.
elim H5.
intros.
clear H0.
clear H3.
clear H5.
apply notE with (alpha H').
unfold alpha.
apply andI.
apply L5.
apply L4 with muddy.
auto.
apply L1 with (fun j:agent =>(In A j)/\(In H j)).
unfold alpha in muddy_H.
```

```
apply andE1 with(BigAnd  (fun j:agent=>
(In A j)/\(In H j->False)) (not_muddy)).
apply muddy_H.
intros.
split.
elim H0.
intros.
exact H3.
apply H6.
elim H0.
intros.
exact H5.
apply L3 with m.
change (Provable (Increase l o)
(BigAnd  (fun j:agent => (In A j)/\(In H' j))
(fun j:agent => (K m ((fun j':agent=>
((muddy j')-->(K m (muddy j'))))j))))).
apply L2.
apply L1 with (fun j:agent=>(In A j)/\( m <> j)).
change(Provable (Increase l o)
((fun i:agent =>(BigAnd (fun j : agent => In A j /\ i <> j)
(fun j : agent => C (muddy j) --> (K i (muddy j)))))m)).
apply BigAndE with (In A).
auto.
auto.
intro.
intros.
split.
elim H0; intros.
auto.
intro.
apply H4.
elim H0;intros.
subst.
exact H8.
apply L1 with (fun i:agent=>(In A i)/\(m = i) \/
(In A i)/\((In H i)->False)).
change(Provable (Increase l (k m))
```

```
(BigAnd (fun j: agent=> ((fun i : agent =>
In A i /\ (m = i))j) \/ (fun i: agent=>
(In A i /\ (In H i -> False)))j)not_muddy)).
apply L7.
auto.
apply BigAndI.
intros.
elim H0;intros.
rewrite <- H5.
unfold not_muddy.
exact H1.
apply L5.
apply L4 with not_muddy.
auto.
unfold alpha in muddy_H.
unfold NotIn in muddy_H.
apply andE2 with(BigAnd (fun j:agent=>(In A j)
/\(In H j)) muddy).
unfold not.
exact muddy_H.
apply L3 with m.
change (Provable (Increase l o)
(BigAnd  (fun g:agent => (In A g)
/\(In H g -> False))
(fun j:agent => (K m ((fun j':agent=>((not_muddy j')
-->(K m (not_muddy j'))))j))))).
apply L2.
apply L1 with (fun j:agent=>(In A j)/\(m <>j)).
change(Provable (Increase l o)
((fun i:agent =>(BigAnd (fun j : agent => In A j /\ i <> j)
(fun j : agent =>
C (not_muddy j) --> (K i (not_muddy j)))))m)).
apply BigAndE with (In A).
auto.
auto.
intros.
split.
elim H0; intros.
```

```
auto.
intro.
elim H0;intros.
apply H8.
subst.
auto.
intros.
elim H0; intros.
elim classic with (j=m).
intros.
left.
subst.
split.
auto.
reflexivity.
trivial.
right.
split.
auto.
intro.
apply H4.
elim H7 with j.
intros.
apply H10.
auto.
auto.
apply KiE with m.
unfold Check_k.
reflexivity.
simpl in |-*.
apply CK.
apply CI.
auto.
change(Provable (Increase (Top.cons o l) c)
((fun G: set=> (Not (alpha G)))H')).
apply BigAndE with (fun G:set=>(Incl A G)
/\(Size_lt (Size H) G)).
apply CE.
```

```
auto.
unfold Check_c.
reflexivity.
simpl in |-*.
auto.
unfold Size_lt.
rewrite <- H2.
split.
unfold Incl.
intros.
apply In_A_m.
constructor.
auto.
Qed.
End Entailment_1.

(*Now we prove that if it is
common knowledge that there are at
least n muddy
children, then, after the announcement that no one
knows whether he's muddy or not, it will be
common knowledge that
there are at least n+1 muddy
children *)

Definition Size_is(n:nat) (G:set):=(Size G)=n.
(*Hypothesis see_muddy: *)

Variable M: level.
Variable  H: set.
Variable m:agent.
Hypothesis In_A_m: forall m: agent,(In A m).
Hypothesis Incl_A_H:forall H: set,(Incl A H).
Hypothesis In_H_m:  forall m: agent, forall H: set,(In H m).


Hypothesis see_muddy: forall l: level, (Provable  l (BigAnd (In A)
(fun i:agent => (BigAnd (fun j:agent=> (In A j)/\(i<>j))
```

```
(fun j:agent=>(C (muddy j)-->(K i (muddy j)))))))).
(* ~i=j -> C(p_i -> K_j p_i)
because the
children
can see each other *)

Hypothesis see_clean:  forall l: level, (Provable  l (BigAnd (In A)
(fun i:agent => (BigAnd (fun j:agent=> (In A j)/\(i<>j))
(fun j:agent=>(C (not_muddy j)-->(K i (not_muddy j)))))))).
(* ~i=j -> C(~p_i -> K_j~p_i)
because the children
can see each other *)

Lemma entailment_2: forall n:nat, forall l: level, forall k: agent,
(Provable  l
(C (BigAnd  (fun G:set =>(Incl A G)
/\(Size_lt (S n) G))
(fun G:set=>(Not (alpha G)))))) ->
(Provable  l (BigAnd  (In A)
(fun j : agent => C (Not (K j (muddy j))
&& Not (K j (not_muddy j)))))) ->
(Provable  l
(C (BigAnd  (fun G:set=>(Incl A G)
/\(Size_lt (S(S n)) G))
(fun G:set=>(Not (alpha G)))))).
intro n.
intro l.
intro.
intro at_least_Sn_muddy.
intro no_one_knows.
apply CI.
auto.
apply L1 with ((fun g:set=>(Incl A g)/\(Size_is (S n) g)
\/(Incl A g)/\(Size_lt (S n) g))).
change(Provable  (Increase l c)
(BigAnd
((fun h: set=> (fun g : set =>
Incl A g /\ (Size_is (S n) g))h \/
```

```
(fun g : set =>Incl A g /\ Size_lt (S n) g)h))
(fun G : set => Not (alpha G)))).
apply L7.
auto.
apply L6.
auto.
apply notI.
apply impI.
intro.
cut(exists G:set,((Incl A G)/\(Size_is (S n) G)) /\
(Provable (Increase(Increase (l) (c)) (o)) (alpha G))).
intros.
elim H1.
intro G'.
intros.
assert (Asp: Provable (Increase l c)
(BigOr (fun g : set => Incl A g /\ Size_is (S n) g) alpha)).
apply Prov.
exact H0.
clear H0.
elim H2;intro.
clear H2.
elim H0; intros.
apply notE  with (BigAnd  (In G') (fun i:agent=>(K i (muddy i)))).
apply BigAndI.
intro m.
intros.
apply entailment_1 with G'.
apply In_A_m.
apply In_H_m.
apply BigAndI.
 intro i.
 intros.
apply BigAndI.
intro j.
intros.
apply CE.
auto.
```

```
unfold Check_c.
reflexivity.
simpl in |-*.
apply Cf.
auto.
change(Provable  (Top.cons o (Top.cons o l)) ((fun a: agent =>(C (muddy a)
 --> (K i (muddy a))))j)).
apply BigAndE with (fun a:agent=>(In A a)/\(i <> a)).
change(Provable (Top.cons o (Top.cons o l))
((fun i:agent=>(BigAnd (fun a : agent => In A a /\ i <> a)
(fun a : agent => C (muddy a) --> (K i (muddy a)))))i)).
apply BigAndE with  (In A).
simpl in |-*.
apply see_muddy.
elim H7.
intros.
auto.
auto.
apply BigAndI.
 intro i.
 intros.
apply BigAndI.
intro j.
intros.
apply CE.
auto.
unfold Check_c.
reflexivity.
apply Cf.
auto.
change(Provable (Decrease (Increase (Increase l c) o)) ((fun a: agent
=>(C (not_muddy a) --> (K i (not_muddy a))))j)).
apply BigAndE with (fun a:agent=>(In A a)/\(i <> a)).
change(Provable  (Decrease (Increase (Increase l c) o))
((fun i:agent=>(BigAnd (fun a : agent => In A a /\ i <> a)
 (fun a : agent => C (not_muddy a) --> (K i (not_muddy a)))))i)).
apply BigAndE with  (In A).
apply see_clean.
```

```
elim H7.
intros.
auto.
auto.
apply H4.
auto.
auto.
apply CE.
auto.
unfold Check_c.
reflexivity.
apply Cf.
auto.
auto.
replace (Size G') with (S n).
unfold Decrease.
unfold Remove_o.
unfold EqLevel.
unfold Increase.
apply ass_at with l.
unfold EqLevel.
unfold Remove_o.
reflexivity.
auto.
auto.
apply notI.
apply impI.
intro.
cut (exists a:agent, (In G' a)).
intro.
elim H6. intro i'. clear H6.
intros.
apply notE with (K i' (muddy i')).
change(Provable
 (Increase l c)
 ((fun i:agent=>(K i (muddy i)))i')).
apply BigAndE with (In G').
apply Prov.
```

```
auto.
auto.
apply andE1 with (Not (K i' (not_muddy i'))).
change(Provable
 (Increase l c)
((fun x:agent=>(Not (K x (muddy x))
 && Not (K x (not_muddy x))))i')).
apply BigAndE with (In A).
apply L8.
auto.
apply ass_at with l.
unfold Increase.
unfold EqLevel.
unfold Remove_o.
reflexivity.
apply no_one_knows.
apply H0.
auto.
cut (Size_is (S n) G').
case G'.
unfold Size_is.
unfold Size.
intro.
inversion H6.
intro i.
intros.
exists i.
compute.
left;trivial.
assumption.
change(exists G : set,
  ((fun G':set=>(Incl A G' /\ Size_is (S n) G'))G) /\
  Provable (Increase (Increase l c) o) (alpha G)).
apply BigOrE.
simpl in |-*.
apply CE.
auto.
unfold Check_c.
```

```
reflexivity.
apply ass_at with l.
unfold Increase.
unfold EqLevel.
unfold Remove_o.
reflexivity.
apply CI.
auto.
apply Prov.
auto.
apply CE.
auto.
unfold Check_c.
reflexivity.
simpl in |-*.
apply ass_at with l.
unfold Increase.
unfold EqLevel.
unfold Remove_o.
reflexivity.
auto.
intros.
elim H0.
intros.
inversion H2.
left.
unfold Size_is.
split.
assumption.
trivial.
right.
unfold Size_lt.
unfold lt.
split.
assumption.
assumption.
Qed
```

.