

Теоретические основы информатики

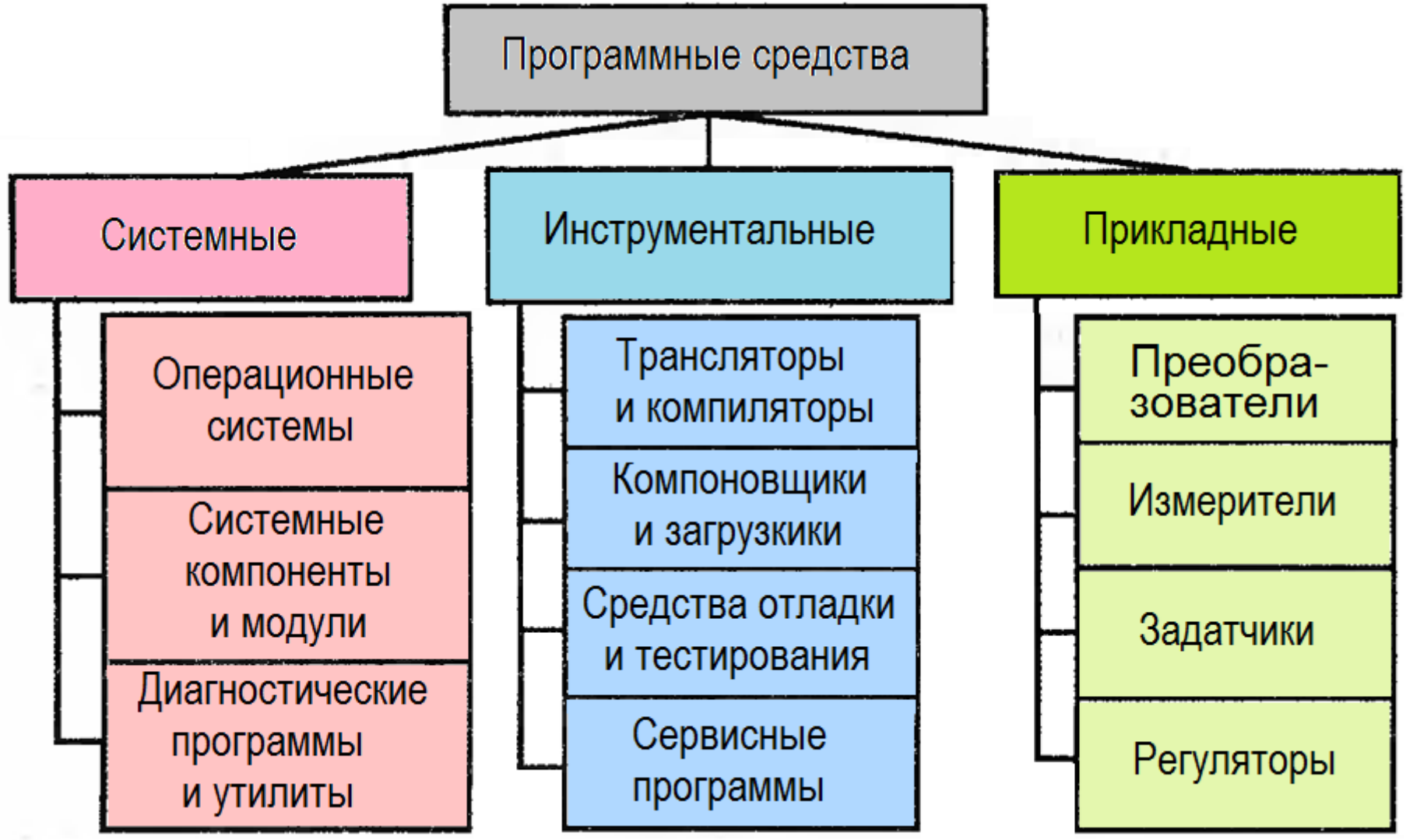
Лекция L04
«Программные средства»

http://vykhovanets.ru/course56/

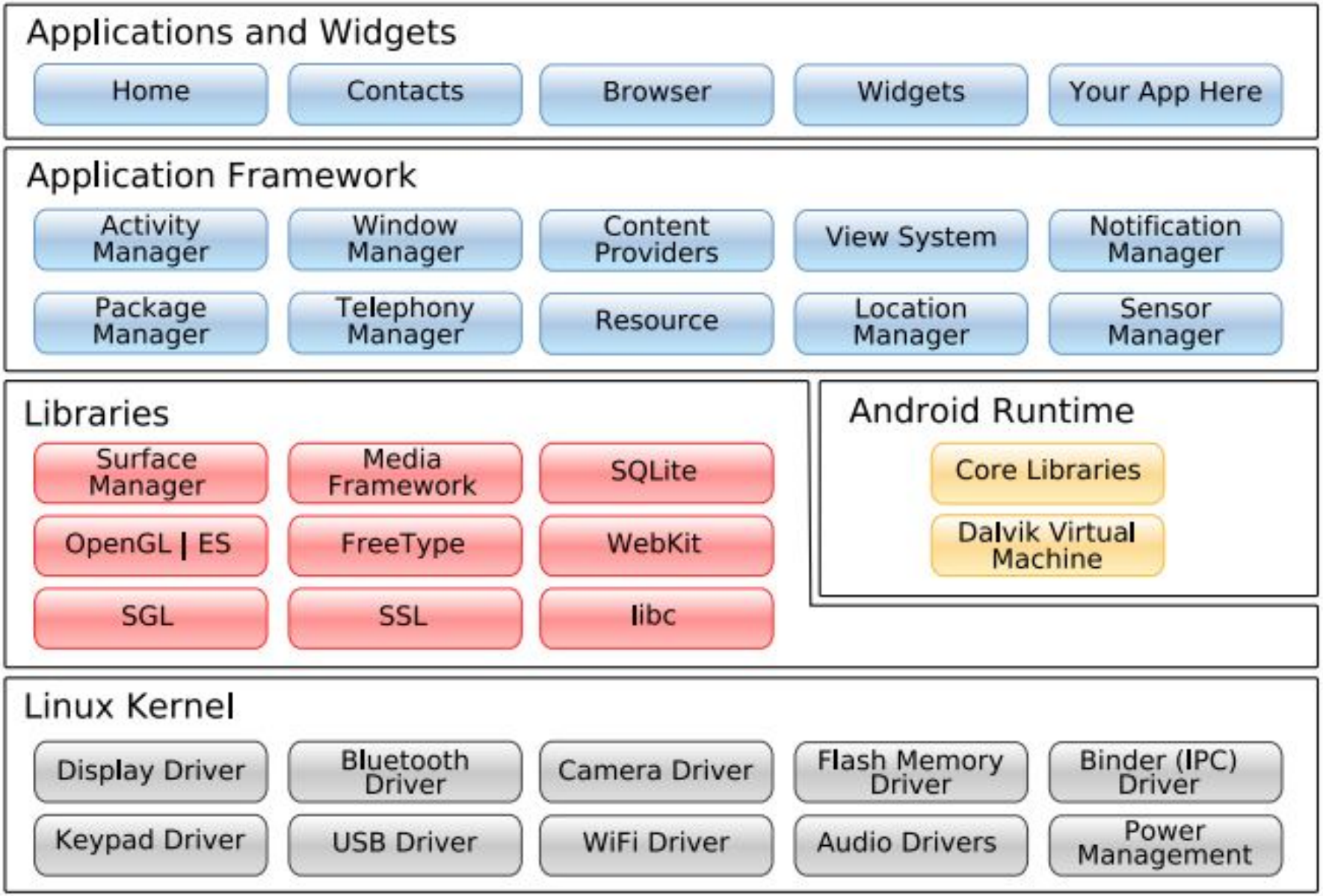
Термины и определения

- **Программа** (program) - данные, предназначенные для управления интерпретаторами (процессорами) в целях реализации определенного алгоритма.
- **Программное средство** – совокупность программ и документов, необходимых для эксплуатации этих программ.
- **Программное обеспечение** (software) – совокупность программных средств и данных, необходимых для их интерпретации.

Программные средства

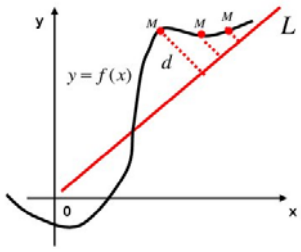


Программное обеспечение



Жизненный цикл



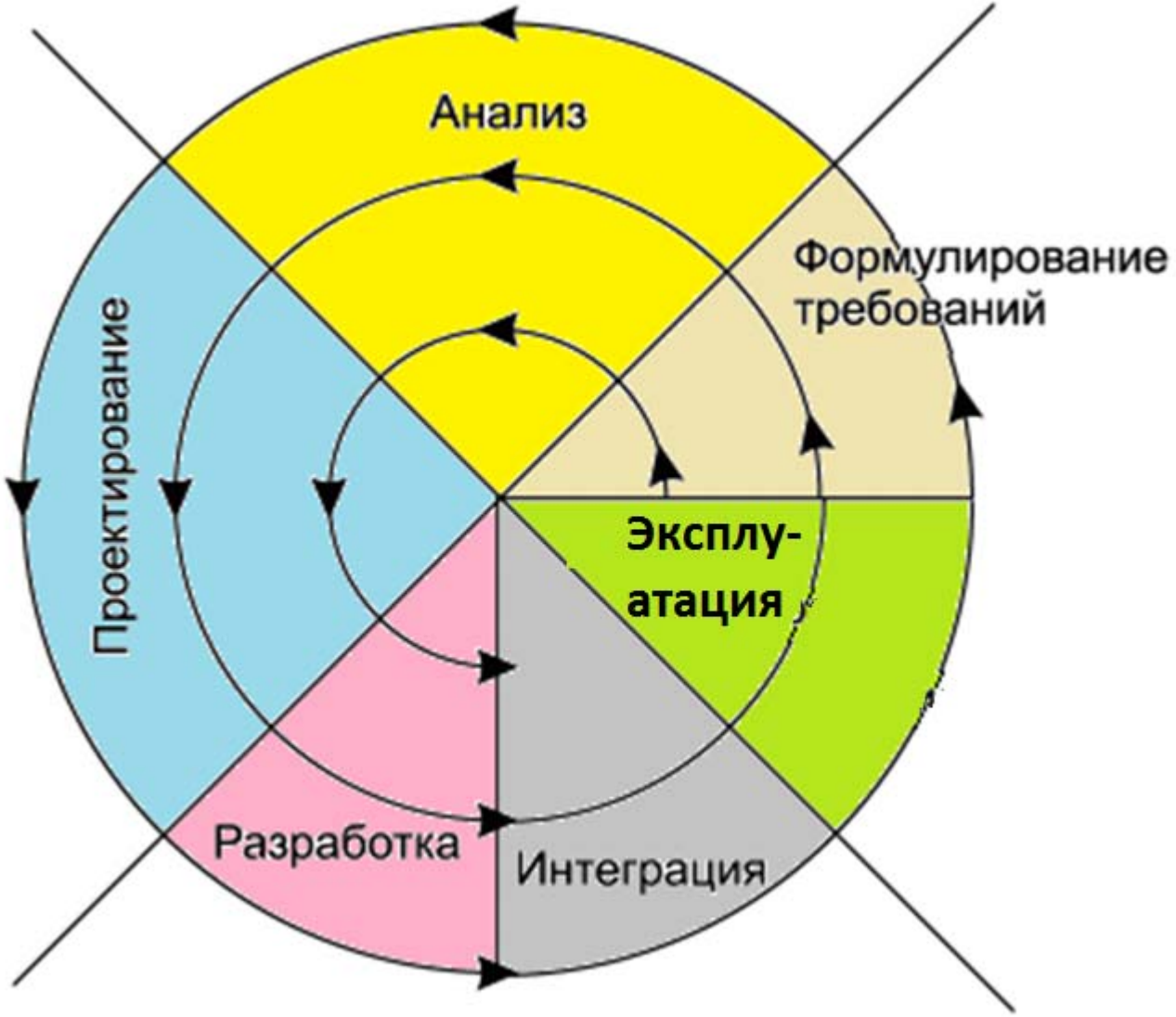








Разработка программ









Свойства программ

- Нельзя полностью протестировать
- Нельзя доказать правильность
- Необходимость сертификации
- Имеется семантический разрыв









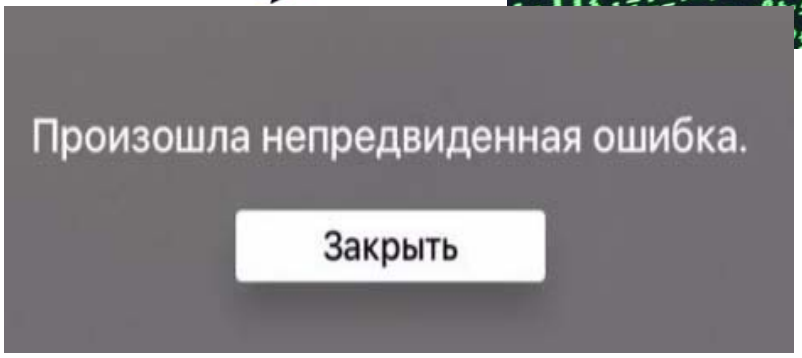
Ошибки в программах

bug – устойчивый неверный результат
error – отказ, воспроизводимая ошибка
fail – сбой, невоспроизводимая ошибка
feature – недокументированный результат
glitch – «непонятный» результат



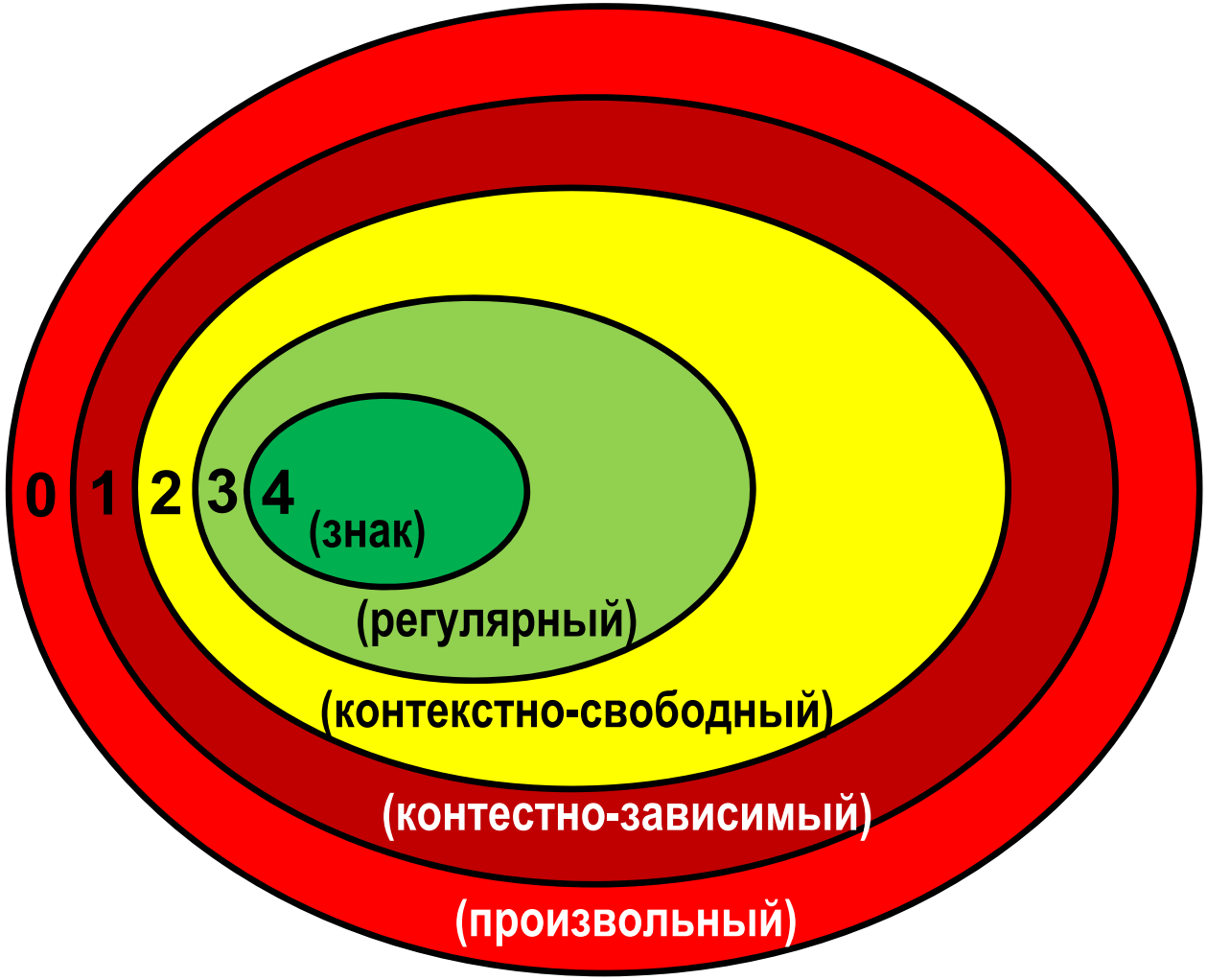








Неразрешимость языков



Классификация по Хомскому:

4 – {a, b, c};

3 – a*(b | c+)?;

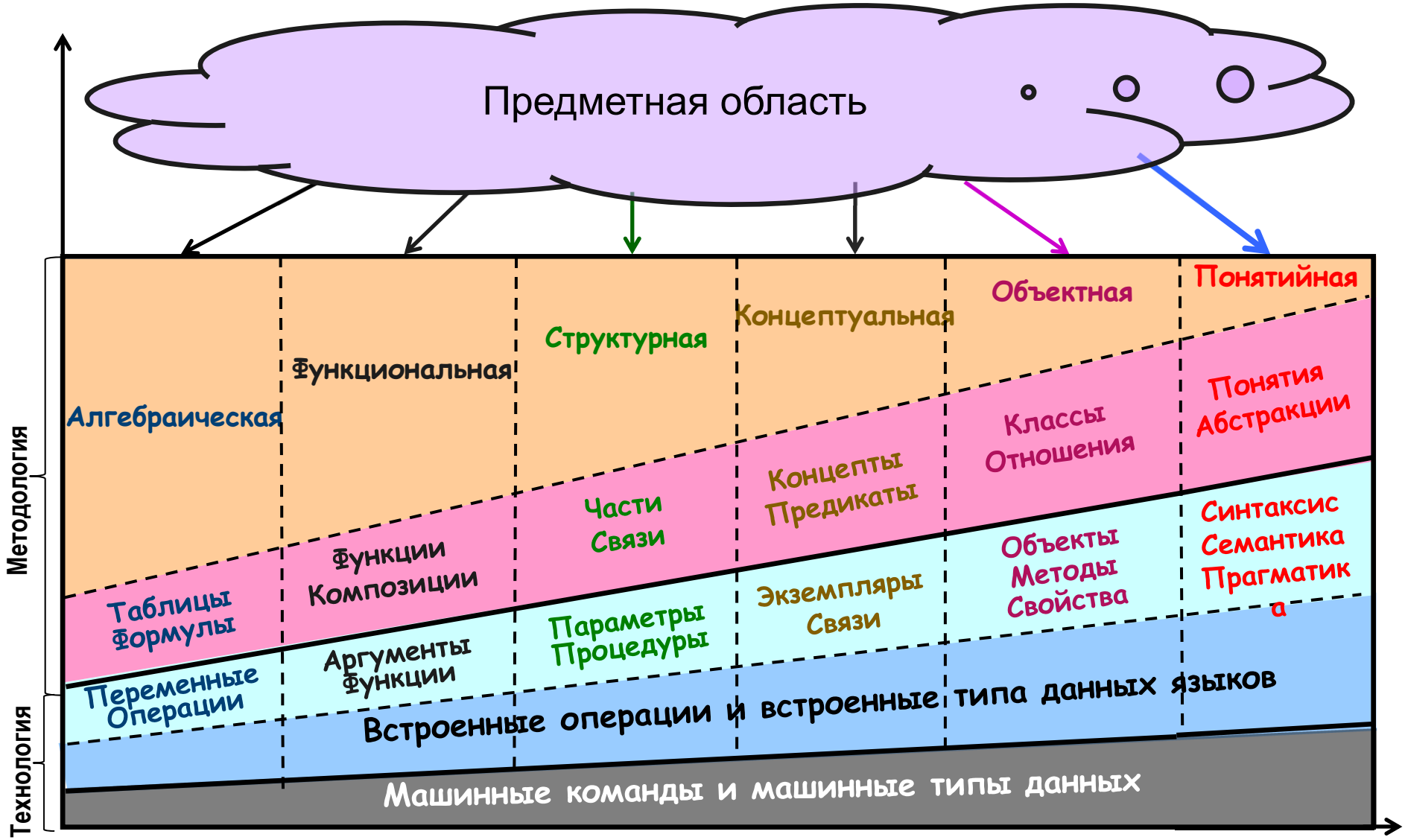
2 – логика высказываний;

1 – языки программирования;

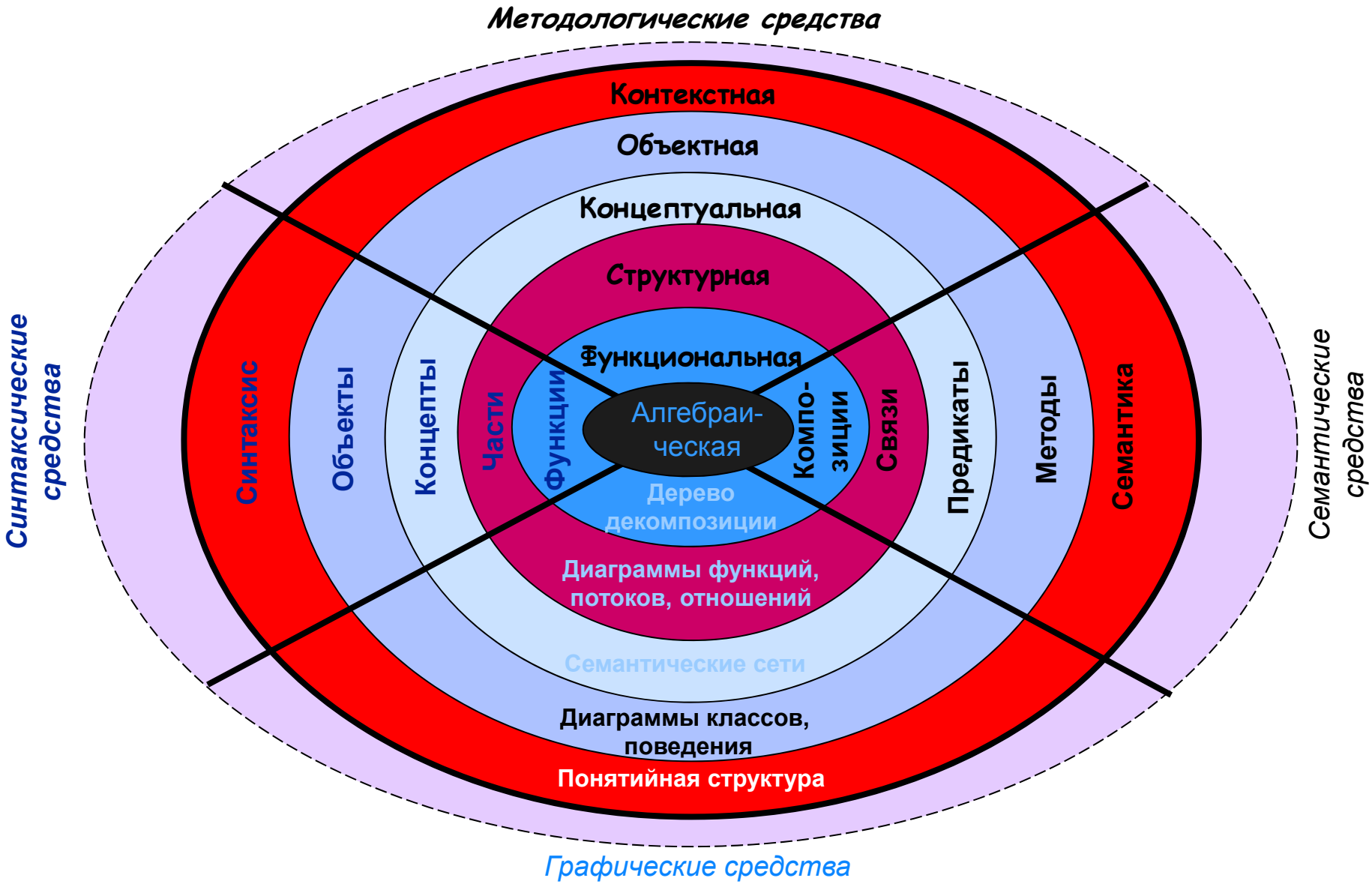
0 – естественный язык.

Неразрешимы языки 0 и 1: никакое нетривиальное свойство программы на этих языках не является алгоритмически разрешимым.

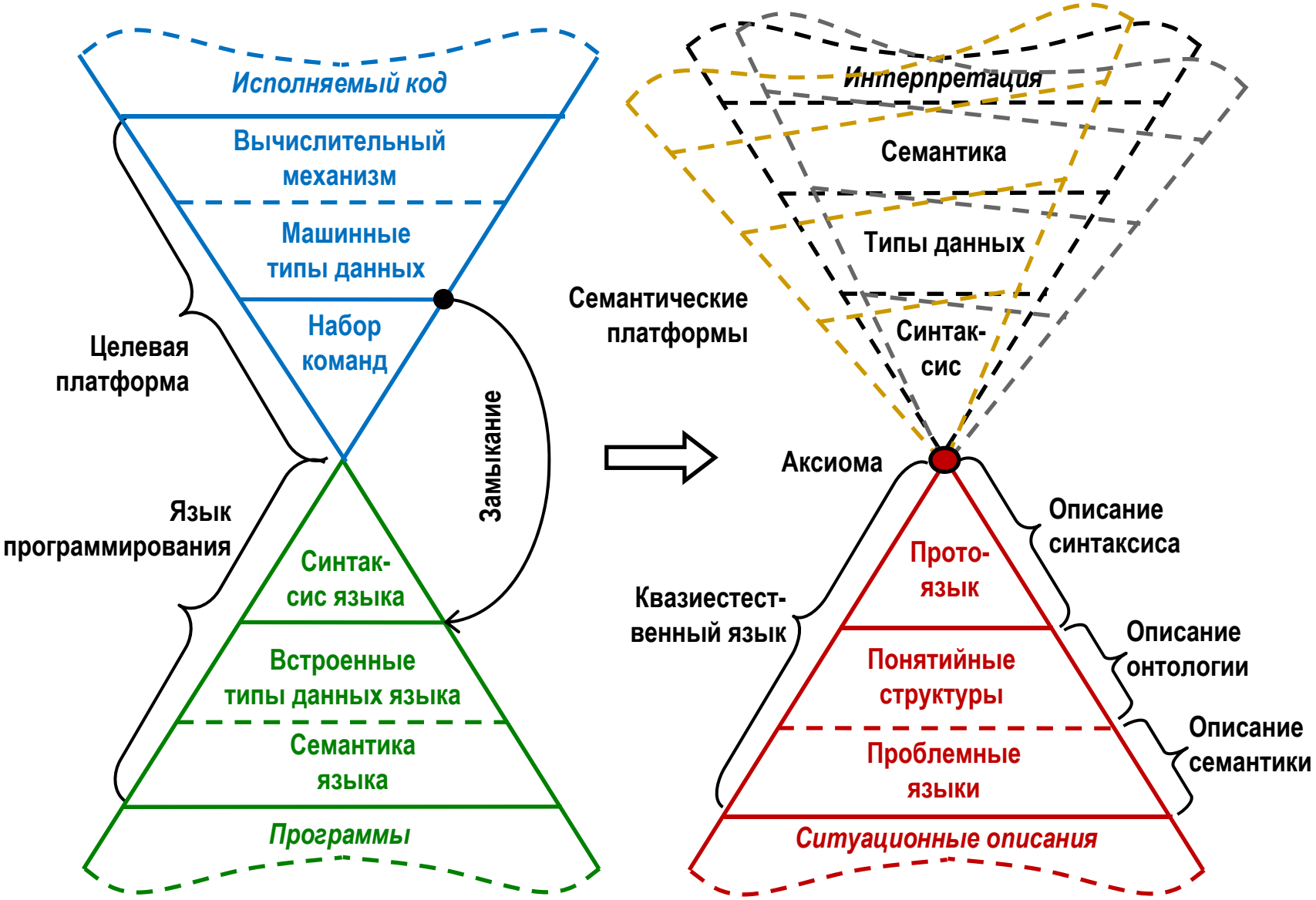
Семантический разрыв



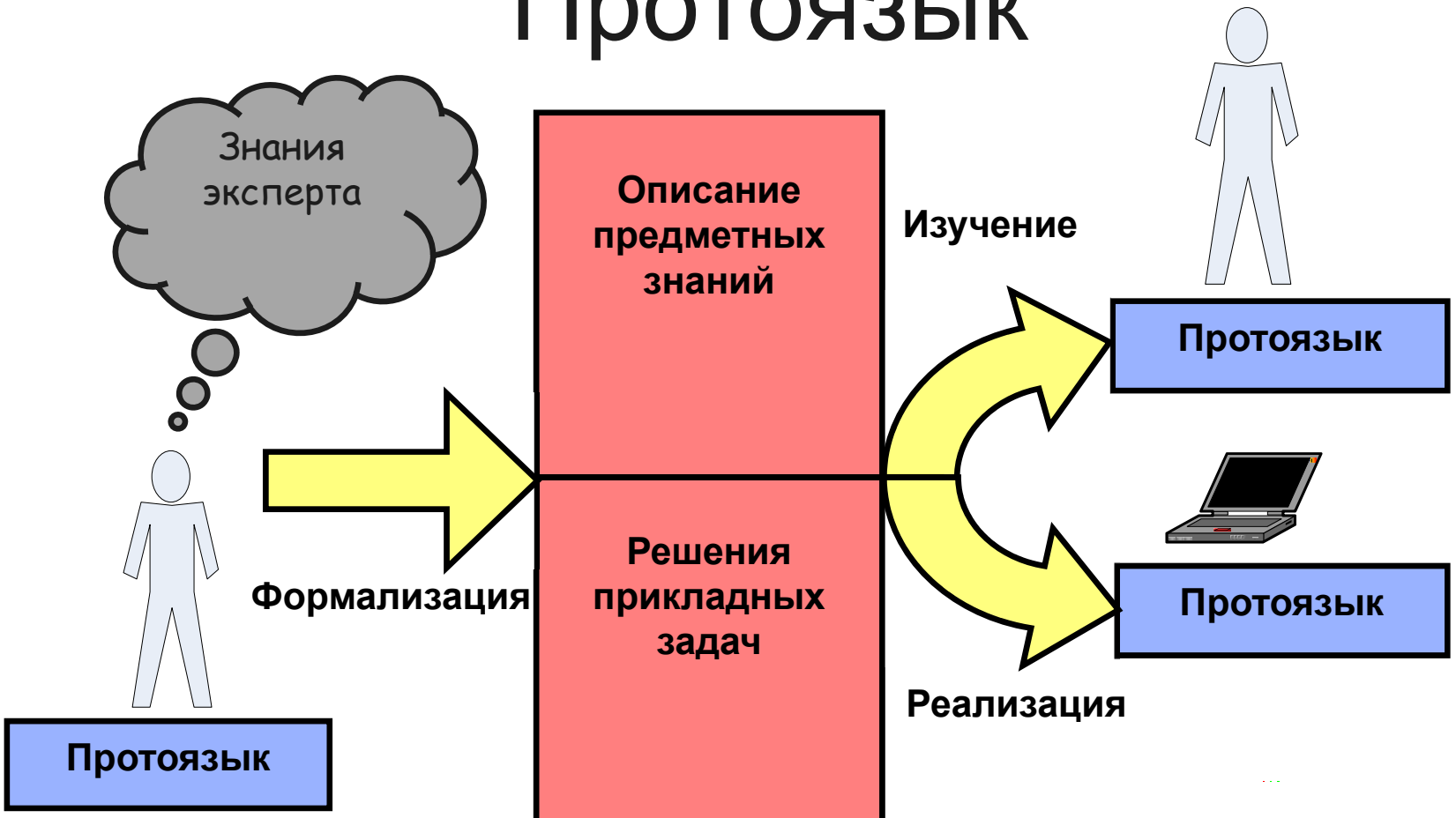
Методологии и технологии

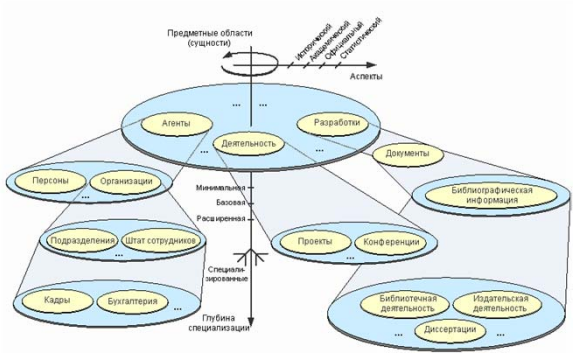


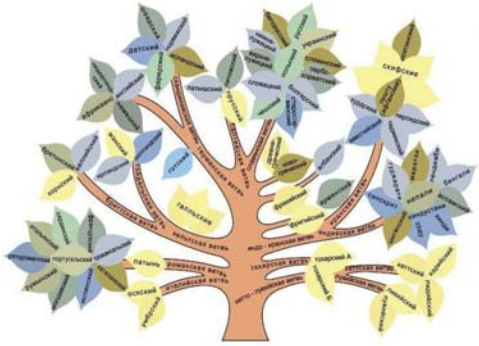
Контекстная технология

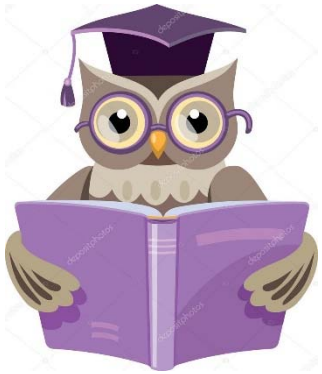


Протоязык


Знания эксперта
Описание предметных знаний
Решения прикладных задач
Формализация
Изучение
Реализация
Протоязык
Протоязык
Протоязык


Предметные области (сущности)
Аспекты
Агенты
Разработки
Деятельность
Документы
Персоны
Организации
Библиографическая информация
Подразделения
Штат сотрудников
Проекты
Конференции
Кадры
Бухгалтерия
Библиотечная деятельность
Издательская деятельность
Диссертации







Пример

Переменная
- () **Var** ()
- " [A-Z][a-z]* " { переменное суждение }

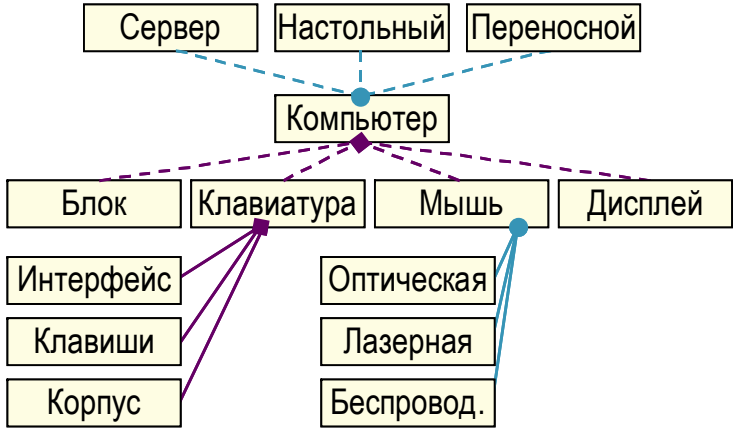
Высказывание
- (Var) **Prop** ()
- 'false | true' { суждение-константа }
- '¬' Prop { суждение-отрицание }
- Prop '→' Prop { суждение-импликация }

Предикат
- (Prop) **Pred** ()
- '(' Pred ')' { суждение в скобках }
- ' ∃|∀ ' Var Pred { суждение-предикат }

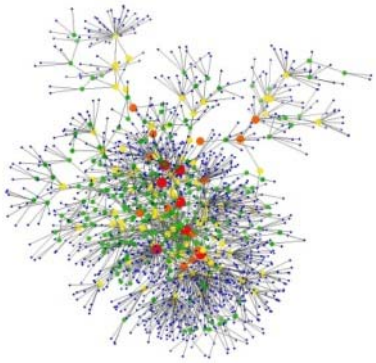
Пустое понятие
- ()()
- Var '←' Pred { умозаключение }

Прикладные знания
- <
- A ← false B ← true C ←(A → ¬B)
- E ← ∀D C → D
- >

Протоязык. Онтология



model → essences [model]
essences → differenciation ***notion***
integration [intension]
differenciation → '**(**' [notions] '**)**'
integration → '**(**' [notions] '**)**'
notions → **notion** [notions]



() ***Сервер*** () () ***Настольный*** () () ***Переносной*** ()
() ***Интерфейс*** () () ***Корпус*** () () ***Клавиши*** ()
() ***Оптическая*** () () ***Лазерная*** () () ***Беспроводная*** ()
() ***Блок*** () () ***Дисплей*** ()
(**Оптическая Лазерная Беспроводная**) ***Мышь*** ()
() ***Клавиатура*** (**Интерфейс Клавиши Корпус**)
(**Сервер Настольный Переносной**) ***Компьютер***
(**Блок Клавитура Мышь Дисплей**)

Протоязык. Синтаксис

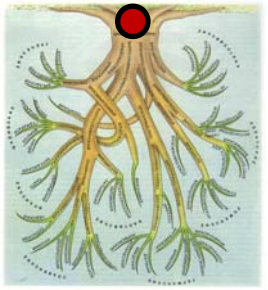



```
cognition  → model [situation] [cognition]
model      → essences [model]
essences   → differenciation notion
                   integration [intension]
intension  → sentence [intension]
sentence   → syntax semantic
syntax     → item [syntax]
item       → notion | '"' [terms] '"'
semantic   → pragmatic [semantic]
pragmatic  → [aspect] '{' [text] '}'
text       → phrase [text]
phrase     → terms | [aspect] '{' text '}'
situation  → [aspect] '<' [text] '>'
```

```
() Boolean ()
    'false' {...}
    'true' {...}
    "[A-Za-z][A-Za-z0-9]*" {...}
    '(' Boolean ')' { }
    'not' Boolean {...}
    Boolean 'and' Boolean {...}
    Boolean [a] 'or' Boolean [b]
        { not (not a and not b) }
< (not x or y) and z >
fuzzy < not (x or y) and z >
```

Протоязык. Аксиома

«Пустые» квадратные скобки передают семантическому интерпретатору тот синтаксический элемент проблемного языка, после которого они применены.



Машинные коды

```
() ()
    '#' "[0-9A-F][0-9A-F]" [ ] { }
() Boolean ()
    'not' Boolean
        { #58 #F7 #D0 #50 }
    Boolean 'or' Boolean
        { #58 #5A #0B #C2 #50 }
```

Язык ассемблера

```
() ()
    '`' "[0-9A-Za-z\s.,]+" [ ] '`' { }
() Boolean ()
    'not' Boolean
        { `pop eax` `not eax`
          `push eax` }
    Boolean 'or' Boolean
        { `pop eax` `pop edx`
        `or eax, edx` `push eax` }
```

Протоязык. Семантика

Семантика определяется в процессе описания проблемного языка и средствами этого языка:

– база индукции – первичные семантические категории, которые непосредственно реализуются (понимаются) семантическим интерпретатором и декларируются перед использованием с помощью аксиомы;

– предположение индукции – все ранее определенные семантические категории, выраженные на проблемном языке;

– индуктивный переход – описание семантики новой или уже существующей семантической категории на уже определенном до этого проблемном языке;

– заключение индукции – определение новой или доопределение существующей семантической категории.



'*not*' Boolean { #58 #F7 #D0 #50 }

Boolean '*or*' Boolean { #58 #5A #0B #C2 #50 }

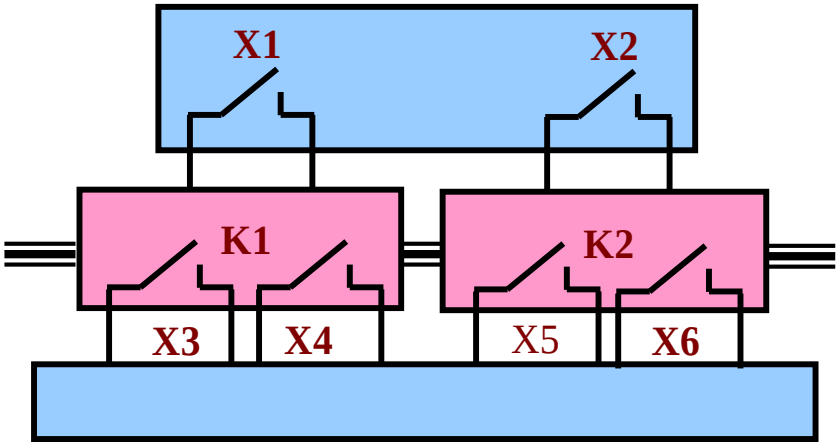
Boolean [a] **'*imp*' Boolean** [b] { not a or b } $\alpha \rightarrow \beta \leftrightarrow \{\neg\alpha \vee \beta\}$

Протоязык. Прагматика



```
( ) Boolean ( )
    'false'
            { Ложь }
            asm { mov eax, 0; push eax }
    'true'
            { Истина }
            asm { mov eax, -1; push eax }
    'not' Boolean [A]
            { Если A Ложь, то Истина, и обратно }
            asm { pop eax; not eax; push eax }
    Boolean [A] 'and' Boolean [B]
            { Если A и B Истина, то Истина, иначе Ложь }
            asm { pop eax; pop edx; and eax, edx; push eax }
    Boolean [A] 'or' Boolean [B]
            { Если A или B Истина, то Истина, иначе Ложь }
            asm { not (not A and not B) }
```

Клапанный механизм



Необходимо реализовать устройство, которое при нажатии кнопки X1 подает управляющий сигнал на открытие клапана K1. После его открытия по сигналу открытого положения X4 снимается управляющий сигнал с клапана K1 и начинает открываться клапан K2.

После открытия K2 по сигналу открытого положения X6 снимается управляющий сигнал с клапана K2 и устройство переходит в устойчивое состояние с открытыми клапанами.

В свою очередь, при нажатии кнопки X2 подается управляющий сигнал на закрытие клапана K2. После его закрытия по сигналу закрытого положения X5 снимается управляющий сигнал с клапана K2 и начинает закрываться клапан K1.

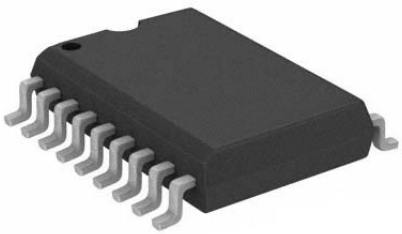
После закрытия K1 по сигналу закрытого положения X3 снимается управляющий сигнал с клапана K1, после чего устройство вновь переходит в устойчивое состояние с закрытыми клапанами.

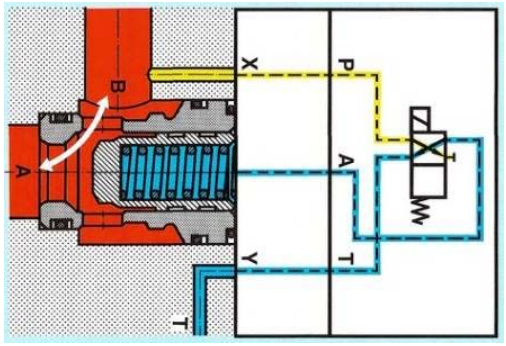


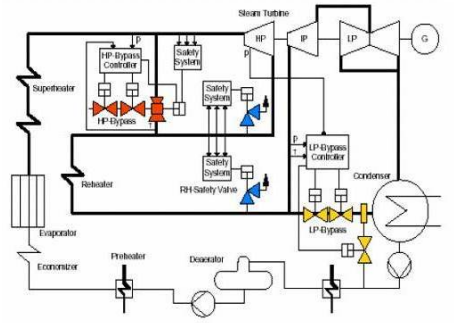




Устройство управления

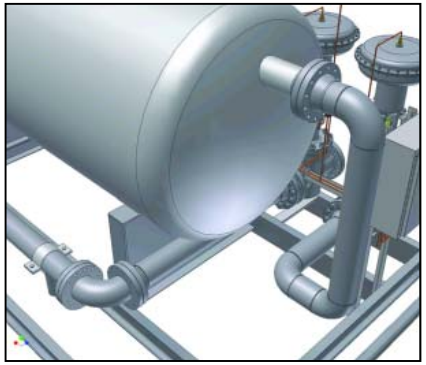


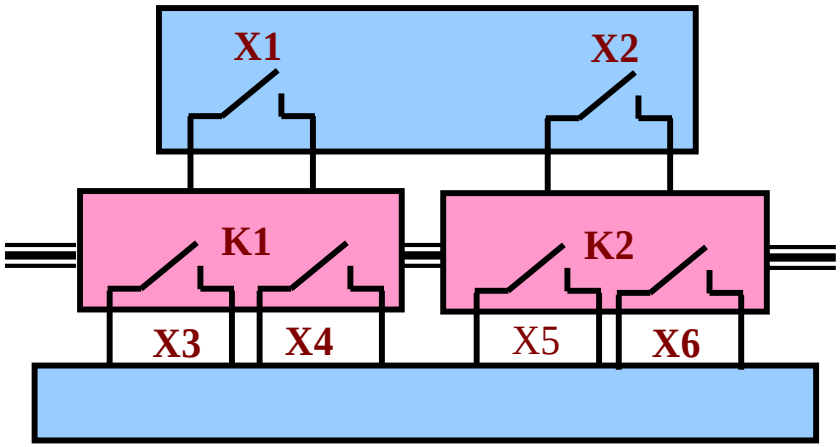





```
1    ( ) ( )
2        '`' "[^`]+" [ ] '`' {}
3    ( ) Сигнал ( )
4        'X1' [`push 0201h;`]{} 'X2' [`push 0202h;`]{}      'X3' [`push 0203h;`]{}
5        'X4' [`push 0204h;`]{} 'X5' [`push 0205h;`]{}      'X6' [`push 0206h;`]{}
6    ( ) Команда ( )
7        'Открытие K1' [`push 0301h;`]{}   'Закрытие K1' [`push 0302h;`]{}
8        'Открытие K2' [`push 0303h;`]{}   'Закрытие K2' [`push 0304h;`]{}
9    ( ) ( )
10       ', ' {`mov eax, 100; L1: inc eax; jnz L1;`}          '. ' { , , }
11       'Выдать' Команда {`pop ebx; mov [ebx], 1;`}
12       'Снять' Команда {`pop ebx; mov [ebx], 0;`}
13       'Ожидать' Сигнал {`pop ebx; L2: mov eax, [ebx]; test eax, 0; je L2;`}
14       'Начало' [`L3: lea eax, L3; push eax;`] '' 'Повторить' [`pop eax; jmp eax;`] {}
```

Программа управления





< Начало.
Ожидать X1, Выдать Открытие K1.
Ожидать X4, Снять Открытие K1, Выдать Открытие K2.
Ожидать X6, Снять Открытие K2.
Ожидать X2, Выдать Закрытие K2.
Ожидать X5, Снять Закрытие K2, Выдать Закрытие K1.
Ожидать X3, Снять Закрытие K1.
Повторить. >



Управление лифтом

1 Ожидание вызова

Если вызовов нет, то ожидать вызов (1). Если вызов со второго этажа, то перейти к открытию дверей (2), иначе – на принятие решения о движении (4).

2 Открытие дверей

Открыть дверь. Если дверь открылась, то перейти на принятие решения о движении (4), иначе повторить открытие двери (2).



3 Закрытие дверей

Закрыть дверь. Если дверь не закрылась, то повторить закрытие двери (3), иначе перейти к определению направления движения (4).

4 Направление движения

Продолжение движения. Если лифт двигался вверх (вниз) и имеются вызовы на движение вверх (вниз), то начать движение вверх (5) (вниз 6). *Изменение направления*. Если вызовов на движение вверх (вниз) нет, но есть вызовы на движение вниз (вверх), то начать движение вниз (6) (вверх 5). *Возврат в начало*. Если вызовов нет и при этом лифт выше (ниже) второго этажа, то начать движение вниз (6) (вверх 5), иначе перейти в состояние ожидания (1).

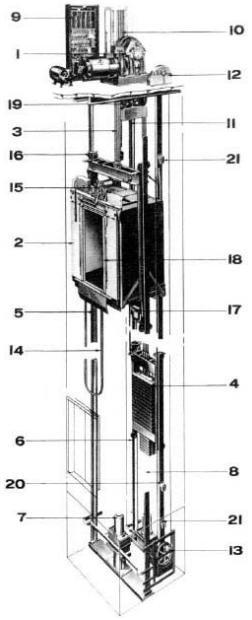
5 Движение вверх

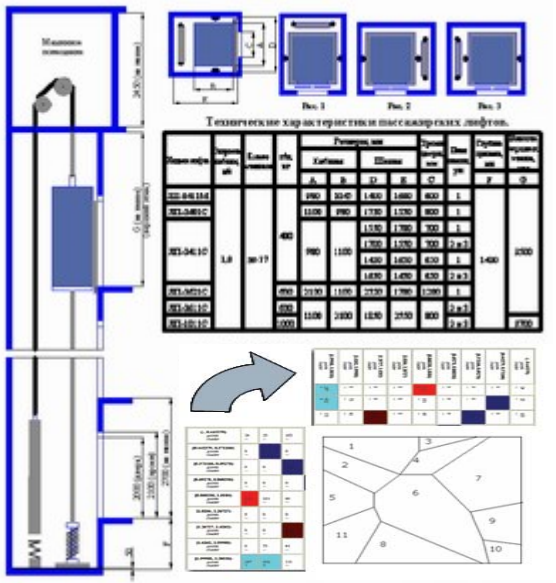
Начать движение вверх. Если при проходе этажа имеется вызов на движение вверх, то остановить лифт и открыть дверь (2), иначе продолжить движение вверх (5).

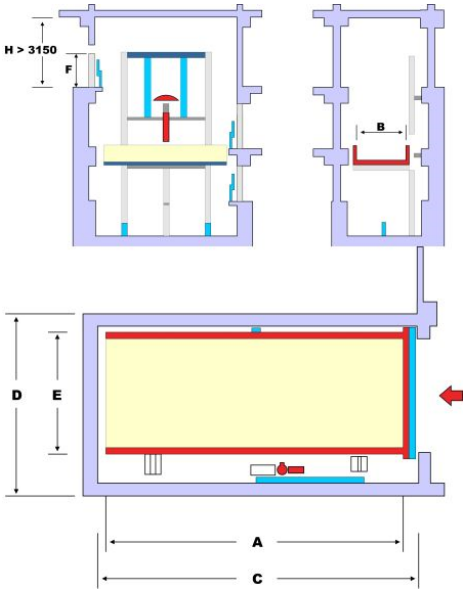
6 Движение вниз

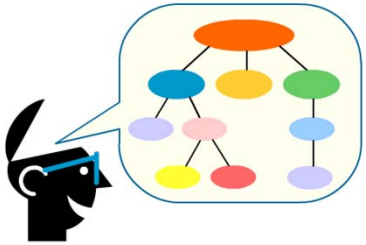
Начать движение вниз. Если при проходе этажа имеется вызов для движения вниз, то остановить лифт и открыть дверь (2), иначе продолжить движение вниз (6).

Понятийная структура









Пустое

Вызов

Логическое

Движение

Дверь

Этаж

Метка

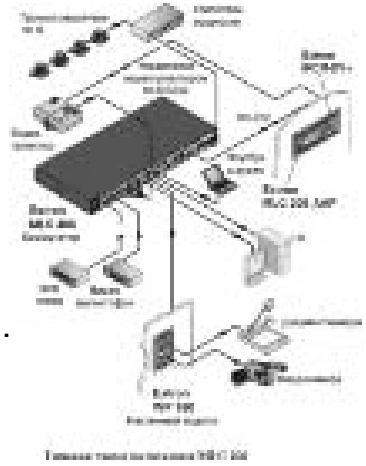
Лифт

Переход

Проблемный язык

```
( ) Движение ( )
    "[Дд]вижение" {…}
( ) Дверь ( )
    "[Дд]верь" {…}
( ) Этаж ( )
    "этажу?" "[0-9]" {…}
    "этажу?" 'вызова' "|вверх|вниз" {…}
    "этажу?" 'лифта' {…}
( ) Вызов (Этаж Движение)
    "вызова?" {…}
( ) Лифт (Этаж Движение Дверь)
    'лифт' {…}
( ) Метка ( )
    "[А-Яа-я][А-Яа-я0-9]*" {…}
( ) Переход ( )
    Метка {…}
( ) Логическое ( )
    Этаж "выше|ниже|равен" Этаж {…
    Вызов "вверх|вниз|нет" {…}
    Дверь "открыта|закрыта" {…}
    'не ' Логическое {}
( ) ( )
    '. ' {…}
    Метка ': ' {…}
    Движение "вверх|вниз|останов" {…}
    Дверь "открыть|закрыть" {…}
    'Если ' Логическое ', ' 'то ' Переход {…}
    'Если ' Логическое ', ' 'то ' Переход ', '
        'иначе ' Переход {…}
```





Ожидание:
- Если вызова нет, то Ожидание.
- Если этаж вызова равен этажу 2, то Открыть, иначе Решение.

Открыть:
- Дверь открыть.
- Если дверь открыта, то Решение, иначе Открыть.

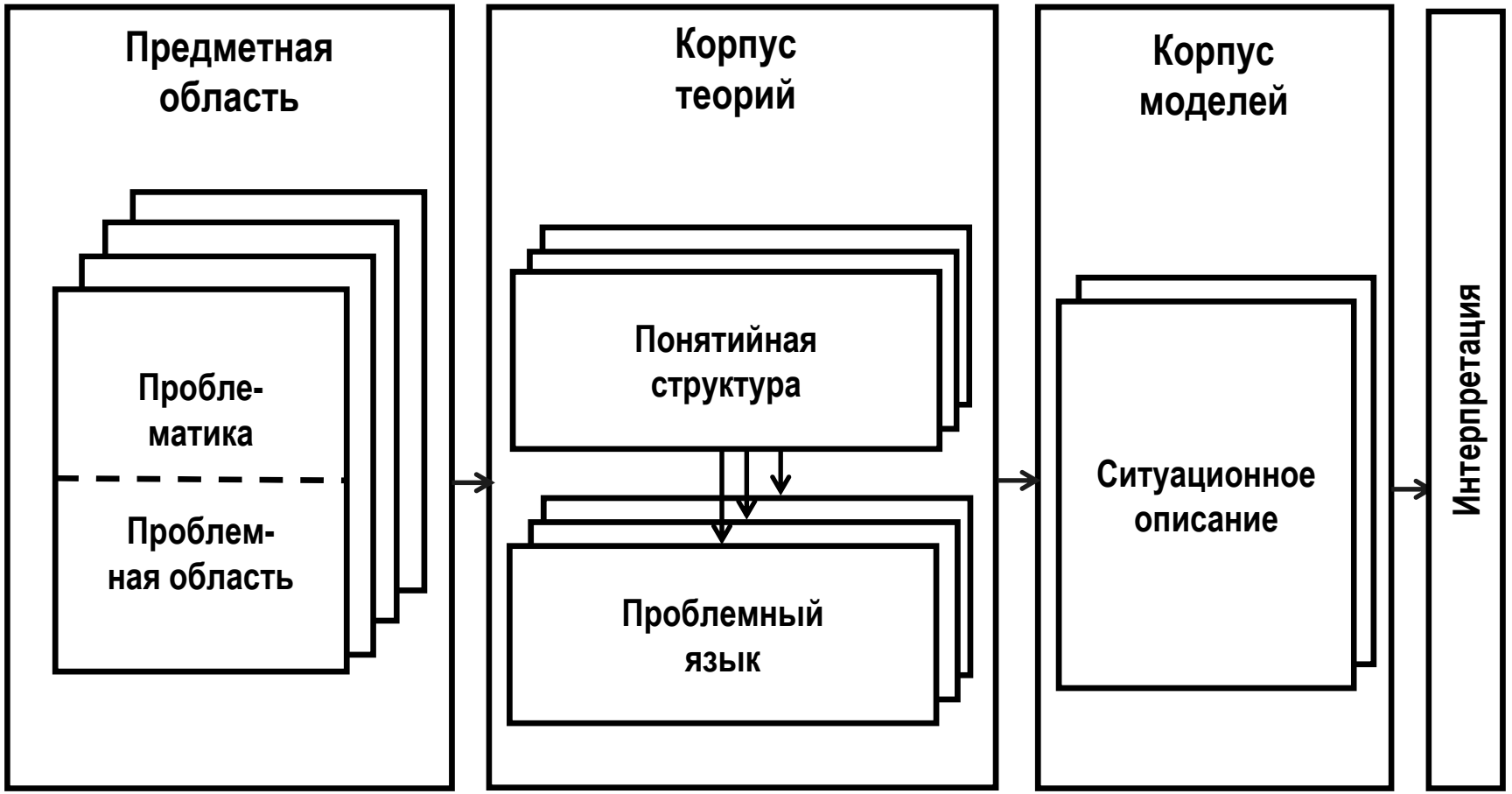
Закрыть:
- Дверь закрыть.
- Если дверь закрыта, то Решение, иначе Закрыть.

Решение:
- Если лифт вверх и вызов вверх, то Вверх.
- Если лифт вниз и Вызов вниз, то Вниз.
- Если не вызов вверх и вызов вниз, то Вниз.
- Если не вызов вниз и вызов вверх, то Вверх.
- Если вызова нет и этаж лифта выше этажа 2, то Вниз.
- Если вызова нет и этаж лифта ниже этажа 2, то Вверх, иначе Ожидание.

Вверх:
- Движение вверх.
- Если этаж лифта равен этажу вызова вверх, то Открыть, иначе Вверх.

Вниз:
- Движение вниз.
- Если этаж лифта равен этажу вызова вниз, то Открыть, иначе Вниз.

Корпусное моделирование



Корпус теорий

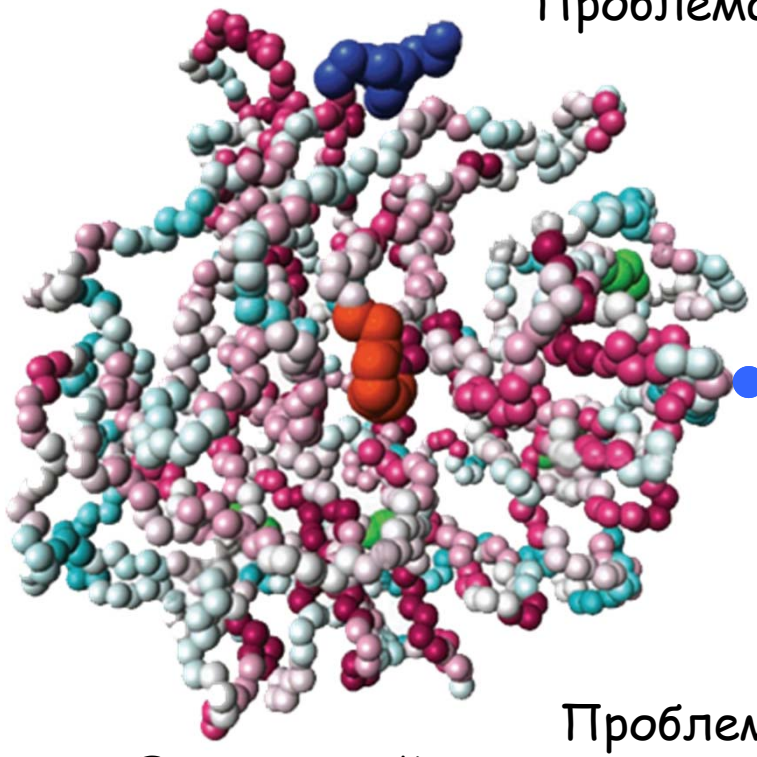
Понятие

Суждение

Умозаключе-
ние

Теория

Корпус

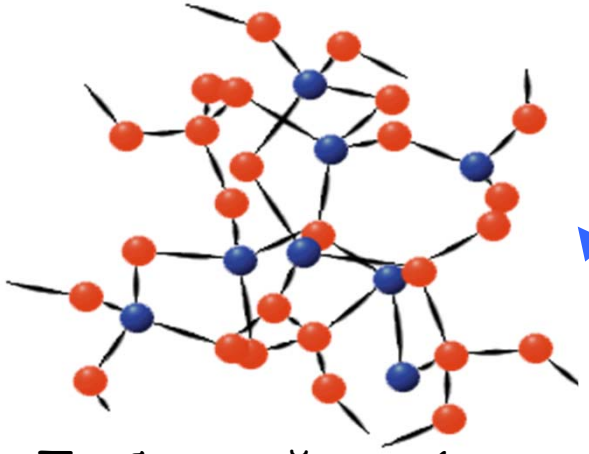


Проблема 1

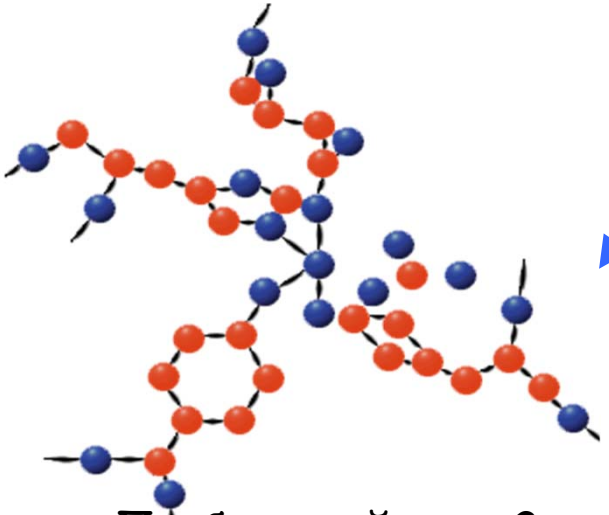
Проблема 2

Естественный язык

(противоречивый, алогичный,
паралогичный)



Проблемный язык 1
(непротиворечивый, логичный)



Проблемный язык 2
(непротиворечивый, логичный)

Противоречивость, паралогичность

План выступления

Тема исследования, научная специальность
Актуальность темы
Цель исследования
Задачи исследования
Методы исследования
Научная новизна
Достоверность результатов
Практическая значимость
Реализация результатов
Апробация работы
Положения, выносимые на защиту
Соответствие паспорту специальности
Публикации
Структура и объем работы
Основное содержание

Литература

1. Непейвода Н.Н., Скопин И.Н. Основания программирования. – Москва-Ижевск: Институт компьютерных исследований, 2003. - 868 с.
2. Липаев В.В. Программная инженерия. Методологические основы. - М.: ТЕИС, 2006. - 608
3. Орлов С. А. Теория и практика языков программирования: Учебник для вузов. СПб.: Питер, 2013. - 688 с.
4. Фаулер М. Предметно-ориентированные языки программирования. – М.: Вильямс, 2012. - 572 с.
5. Выхованец В.С., Иосенкин В.Я. Понятийный ан и контекстная технология программирования // Проблемы управления. - 2005. - № 4. - С. 2-11.

