Трудноформализуемые задачи и контекстная технология программирования

Говоров М.И., Выхованец В.С.

Московский государственный технический университет им. Н.Э. Баумана,
Институт проблем управления им. В.А. Трапезникова РАН

Неформализуемость

A. Существуют классы корректно поставленных задач – массовых проблем, имеющих эффективно распознаваемые условия, для которых доказано отсутствие каких-либо алгоритмов решения:
   – проблемы остановки и эквивалентности алгоритмов;
   – тождество математических выражений;
   – другие проблемы (топология, теория групп и т.д.).
B. Неформализуемые задачи не имеет общего алгоритма решения, применимого ко всем задачам:
   – для решения подклассов задач требуется разработка индивидуальных алгоритмов.
   – для некоторых задач требуется разработка уникальных методов решения;
   – некоторые задачи решения не имеют.

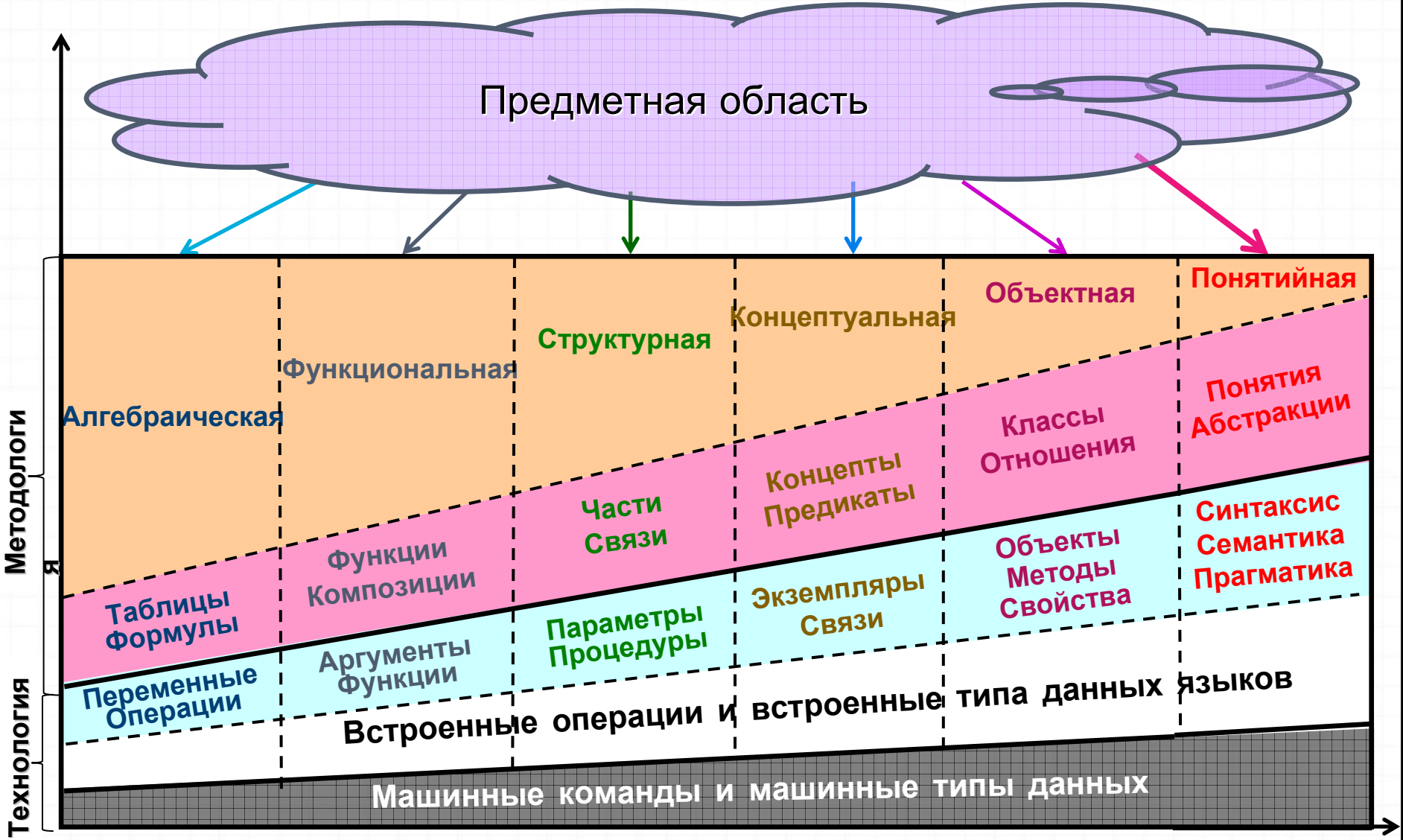
Трудноформализуемость

A. Конкретизация массовой проблемы позволяет находить решение некоторых прикладных задач, но каждый раз по своему. Такие задачи будем называть трудноформализуемыми.
B. Пример – распознавание применимости алгоритма:
   – в общей постановке задача неразрешима;
   – для конкретного алгоритма решение может быть найдено;
   – для другого алгоритма требуется поиск нового решения.
C. Другие примеры трудноформализуемых задач:
   – аннотирование текстов;
   – доказательство теорем;
   – распознавание образов;
   – принятие решений;
   – классификация и кластеризация данных.

Суть подхода

A. Решение трудноформализуемых задач основано на изучении предметной области под углом зрения некоторой проблематики, конкретизирующей неразрешимую проблему до ее разрешимого частного случая.
B. Существующие методологии и технологии программирования порождают труднопреодолимый семантический разрыв между содержательными представлениями о проблемной области и ее формальной моделью на языках программирования.
C. Предлагается метод сокращения семантического разрыва заключающийся в создании для каждой трудноформализуемой задачи своего проблемного языка, являющегося результатом описания онтологии проблемной области, задания способов выражения прикладных понятий и определение их семантики.

Методологии



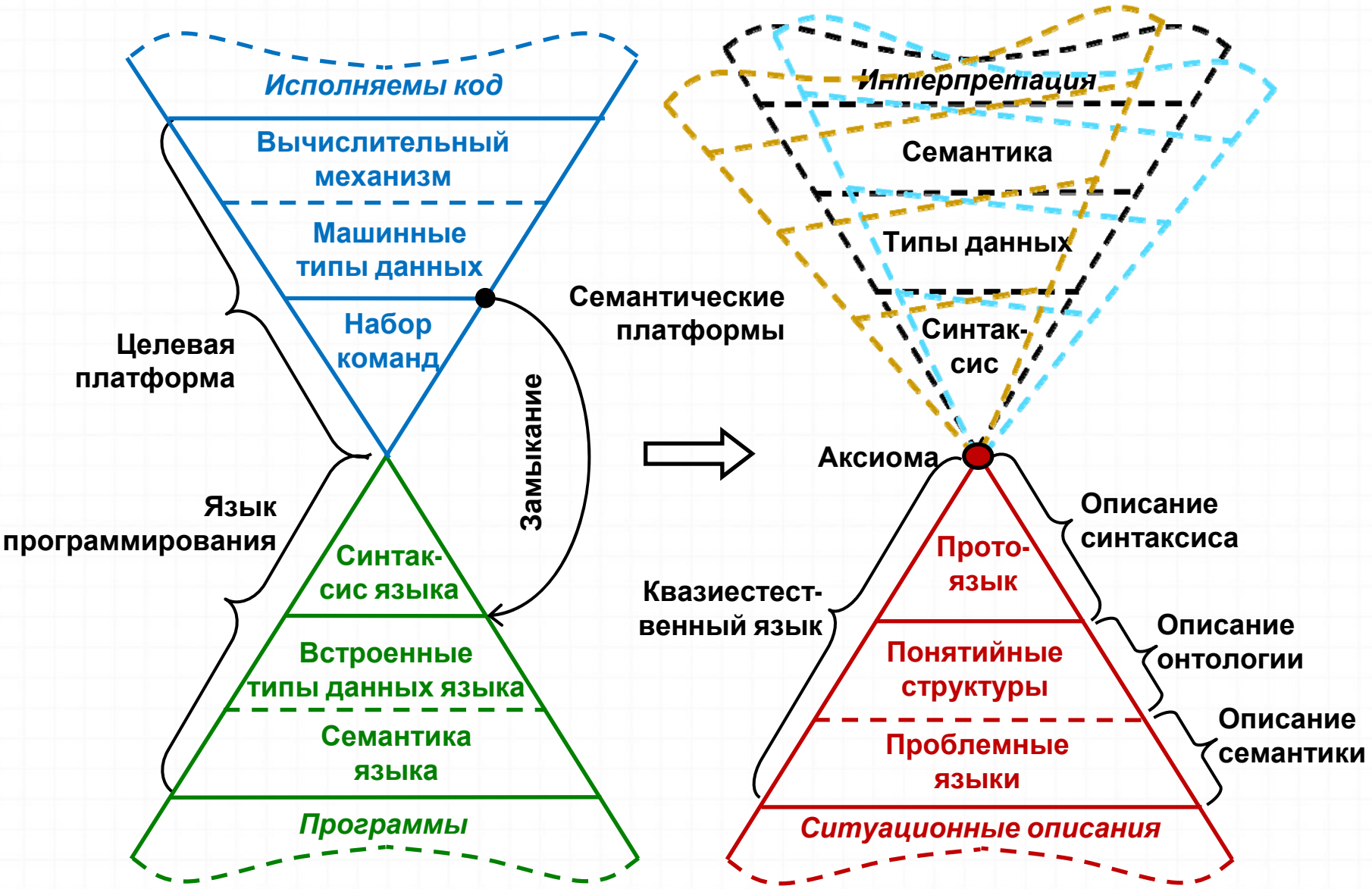
Технологии


Методологические средства
Контекстная
Объектная
Концептуальная
Структурная
Функциональная
Алгебраи-
ческая
Дерево
декомпозиции
Функции
Части
Компо-
зиции
Связи
Диаграммы функций,
потоков, отношений
Семантические сети
Диаграммы классов,
поведения
Понятийная структура
Синтаксис
Объекты
Концепты
Предикаты
Методы
Семантика
Синтаксические
средства
Семантические
средства
Графические средства

Языки

A. Специализированные языки:
   – предметно-ориентированные (FDB, SFC);
   – проблемно-ориентированные (R, SQL, Postscript).
B. Универсальные языки:
   – императивные (C, C++, C#, VB, Java, Perl, Ruby, Python);
   – декларативные (Prolog, Рефал).
C. Метаязыки:
   – регулярные (FORTH, LISP, Tex);
   – бесконтекстные (Рефал, Nemerle, MPS);
   – контекстные (языки контекстной технологии).
D. Протоязыки:
   – дедуктивные (YACC, Bison);
   – индуктивные (протоязык контекстной технологии).

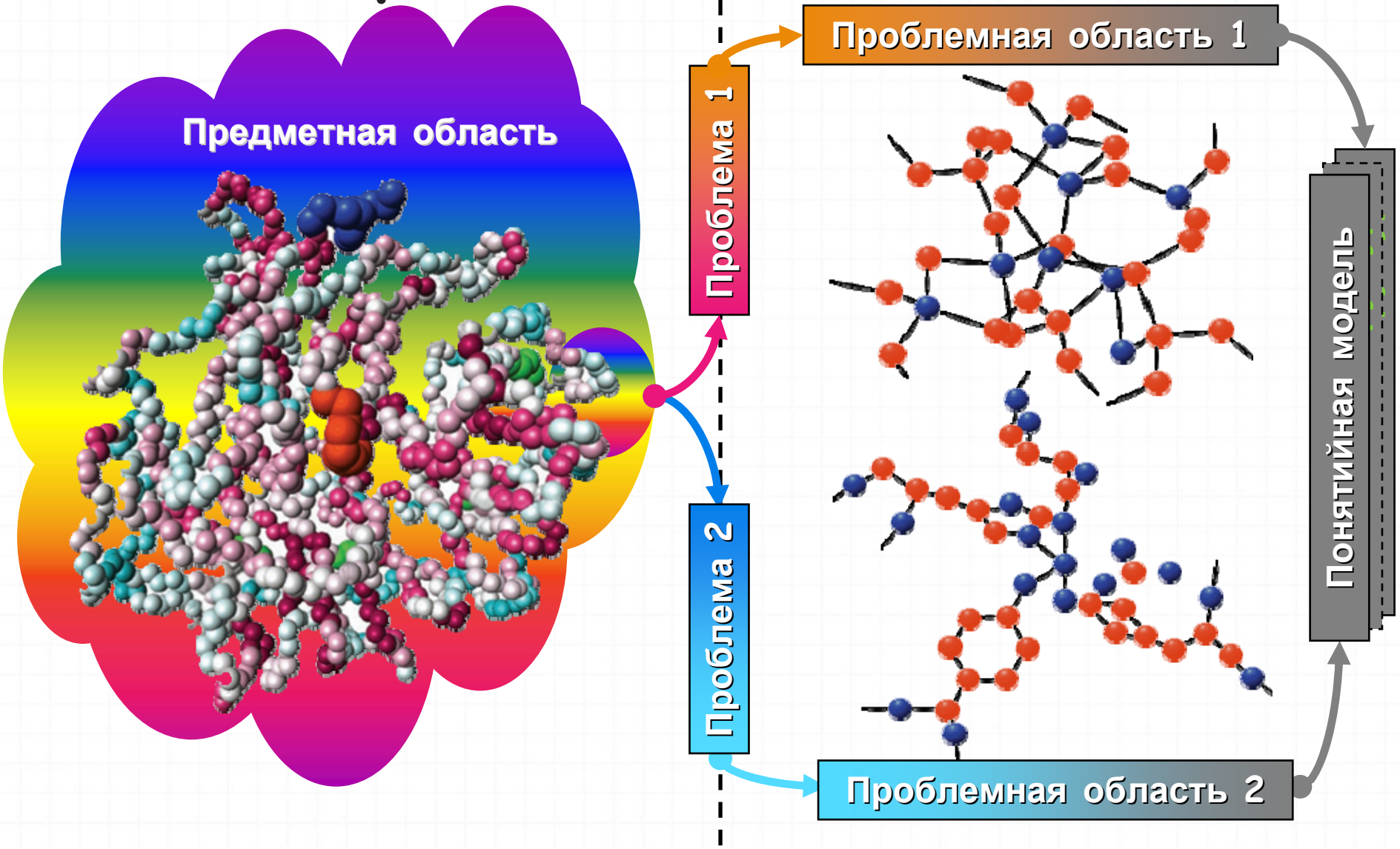
Контекстная технология



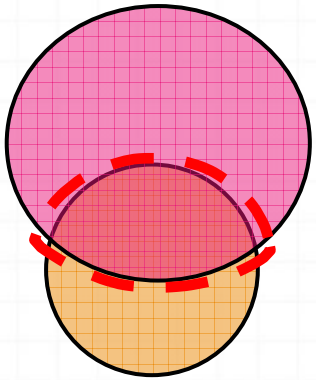
Понятийный анализ

A. Фиксация предметной области и ее проблематизация.
B. Разделение понятий на сигнификативные и денотационные и их означивание.
C. Установление связей абстрагирования денотационных понятий на основе интеграции и дифференциации признаков.
D. Формализация проблемной области путем создания ее понятийной структуры и верификация понятийной структуры путем вычисления схем понятий.
E. Создание понятийной модели путем описания синтаксиса, семантики и прагматики понятий.

Проблематизация



Абстрагирование



3

Настольный

Сервер

Переносной

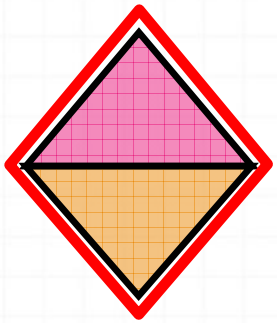
Компьютер

Клавиатура

Интерфейс

Клавиши

Корпус



Обобщение

Агрегация

4

2

Ассоциация

Типизация

Компьютер

Блок

Клавиатура

Мышь

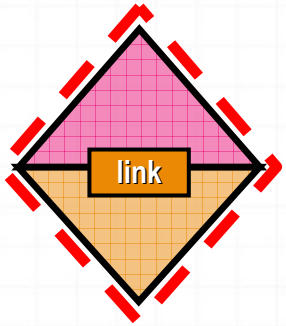
Дисплей

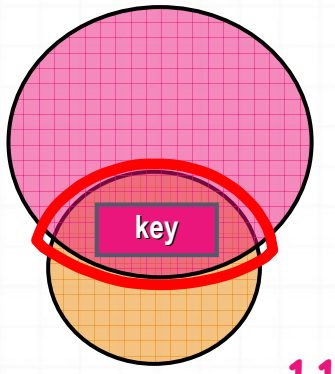
Мышь

Оптическая

Лазерная

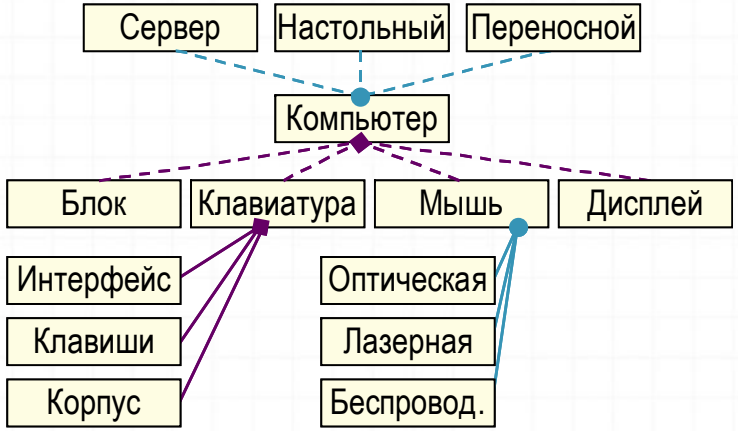
Беспровод.





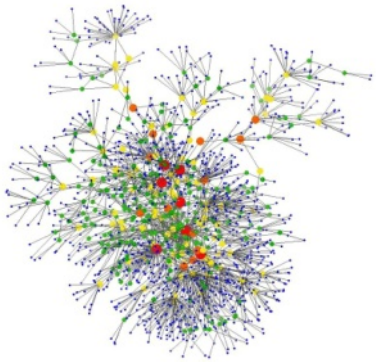
1

Онтология

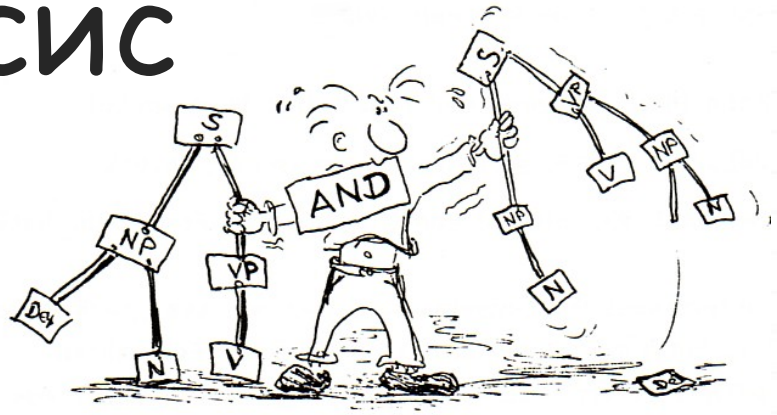


model → essences [model]
essences → differenciation ***notion***
integration [intension]
differenciation → '(' [notions] ')'
integration → '(' [notions] ')'
notions → **notion** [notions]

() ***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**)



Синтаксис

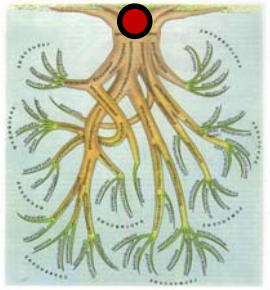


cognition → model [situation] [cognition]
model → essences [model]
essences → differenciation ***notion***
integration [intension]
intension → sentence [intension]
sentence → syntax semantic
syntax → item [syntax]
item → **notion** | '″' [***terms***] '″'
semantic → pragmatic [semantic]
pragmatic → [***aspect***] '**{**' [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 >

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` }
```

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 } $\alpha \rightarrow \beta \leftrightarrow \{\neg\alpha \vee \beta\}$

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) }
```

Символьные вычисления

```
() String ()
        "'.*'" { … }
        String '&' String { … }
() Expression ()
        "[0-9]+" [n] { n } dif { '0' }
        'x' { 'x' } dif { '1' }
        '(' Expression ')' [e]
              { '(' & e & ')' } dif { '(' & dif { e } & ')' }
        '-' Expression [e]
              { '-' & e } dif { '-' & dif { e } }
        Expression [e1] '*' Expression [e2]
              { e1 & '*' & e2 }
              dif { '('& e1 & '*' & dif { e2 } & '+' & dif { e1 } & '*' & e2 & ')' }
        Expression [e1] "+|-" [o] Expression [e2]
              { e1 & o & e2 } dif { '(' & dif { e1 } & o & dif { e2 } & ')' }
```

$C' = 0$

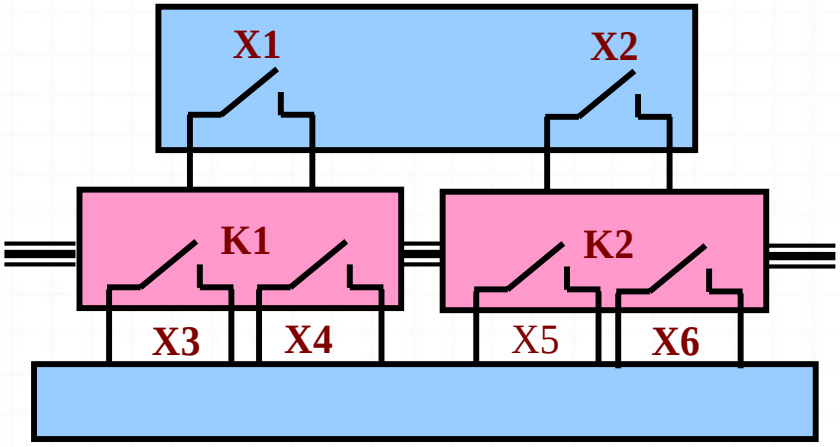
$x' = 1$

$(f+g)' = f' + g'$

$(fg)' = f'g+fg'$

```
dif < -x*x+5*(x–3) >  →  -(x*1+1*x)+(5*(1-0)+0*(x-3))
equ< dif{ -x*x+5*(x–3) } >  →  -2*x+5
```

Клапанный механизм



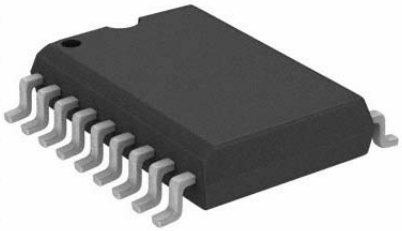
Необходимо реализовать устройство, которое при нажатии кнопки X1 подает управляющий сигнал на открытие клапана K1. После его открытия по сигналу открытого положения X4 снимается управляющий сигнал с клапана K1 и начинает открываться клапан K2.

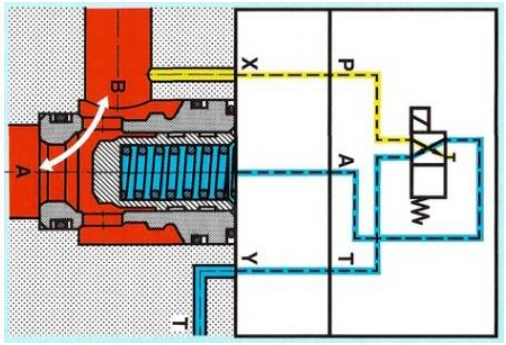
После открытия K2 по сигналу открытого положения X6 снимается управляющий сигнал с клапана K2 и устройство переходит в устойчивое состояние с открытыми клапанами.

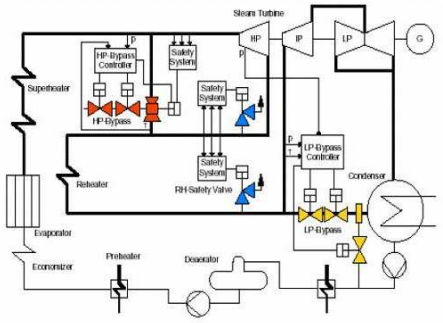
В свою очередь, при нажатии кнопки X2 подается управляющий сигнал на закрытие клапана K2. После его закрытия по сигналу закрытого положения X5 снимается управляющий сигнал с клапана K2 и начинает закрываться клапан K1.

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

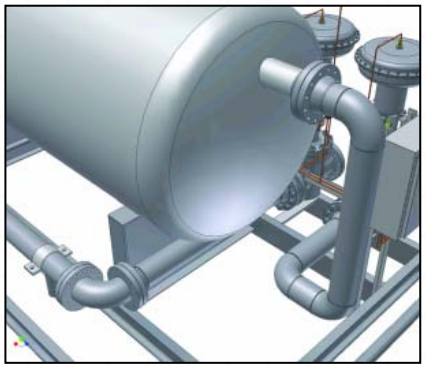


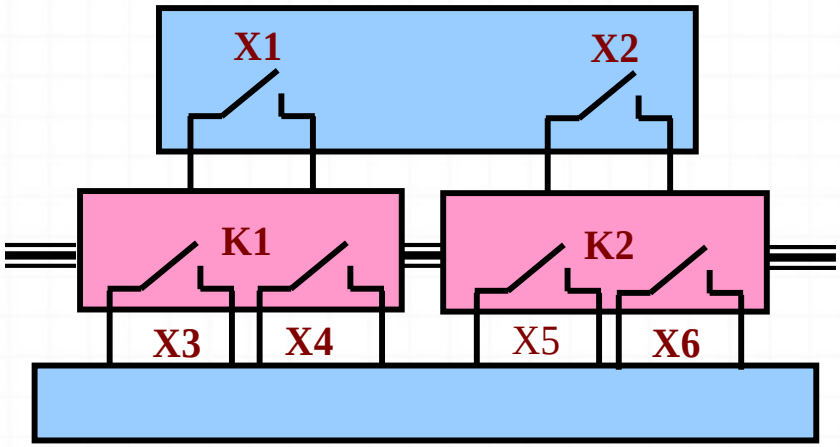





```
1    ( ) ( )
2        '`' "[^`]+" [ ] '`' {}
3    ( ) Сигнал ( )
4        'X1' [`push 0201h;`]{} 'X2' [`push 0202h;`]{}      'X3' [`push 0203h;`]{}
5        'X4' [`push 0204h;`]{} 'X5' [`push 0205h;`]{}      'X6' [`push 0206h;`]{}
6    ( ) Команда ( )
7        'Открытие K1' [`push 0301h;`]{}   'Закрытие 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).

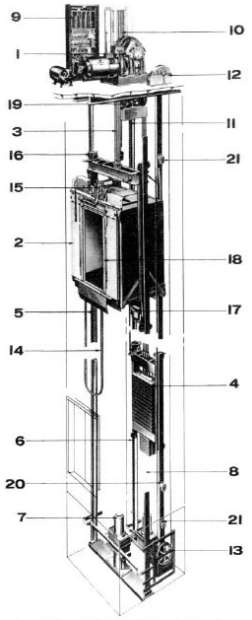
5 Движение вверх

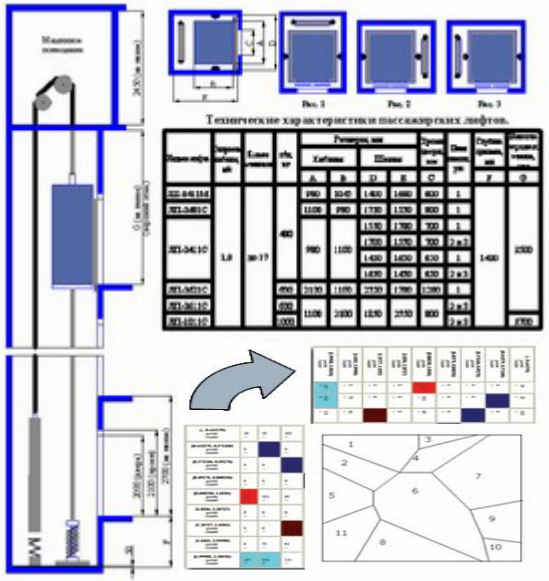
Начать движение вверх. Если при проходе этажа имеется вызов на движение вверх, то остановить лифт и открыть дверь (2), иначе продолжить движение вверх (5).

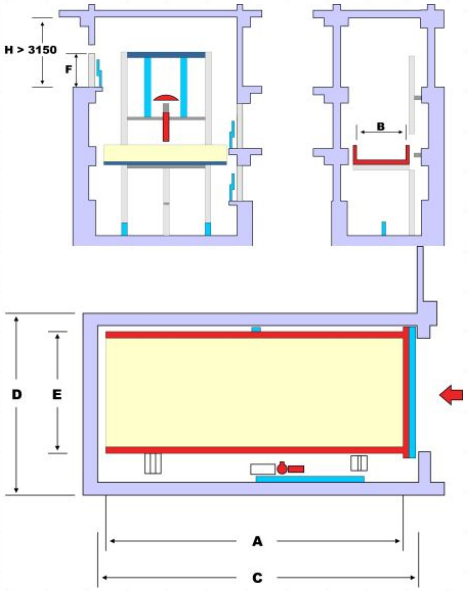
6 Движение вниз

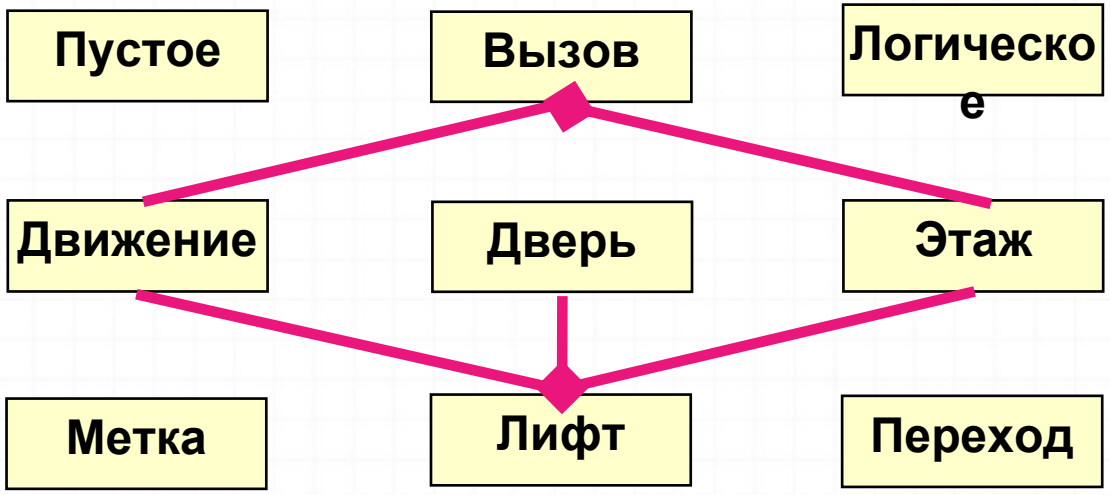
Начать движение вниз. Если при проходе этажа имеется вызов для движения вниз, то остановить лифт и открыть дверь (2), иначе продолжить движение вниз (6).

Предметная онтология



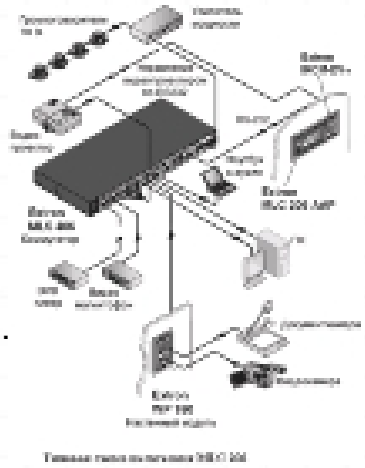






Проблемный язык

```
( ) 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.

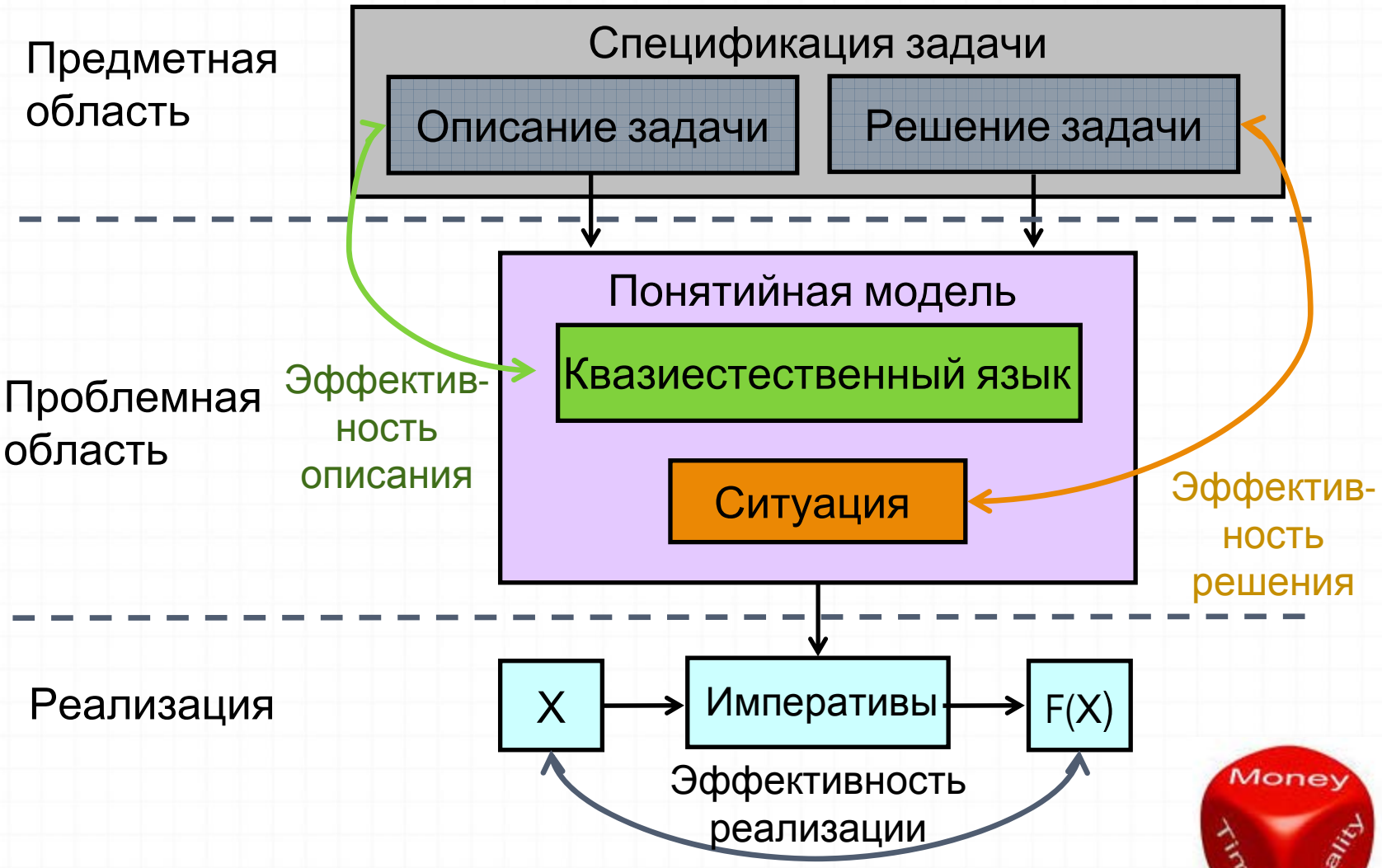
Закрыть:
- Дверь закрыть.
- Если дверь закрыта, то Решение, иначе Закрыть.

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.

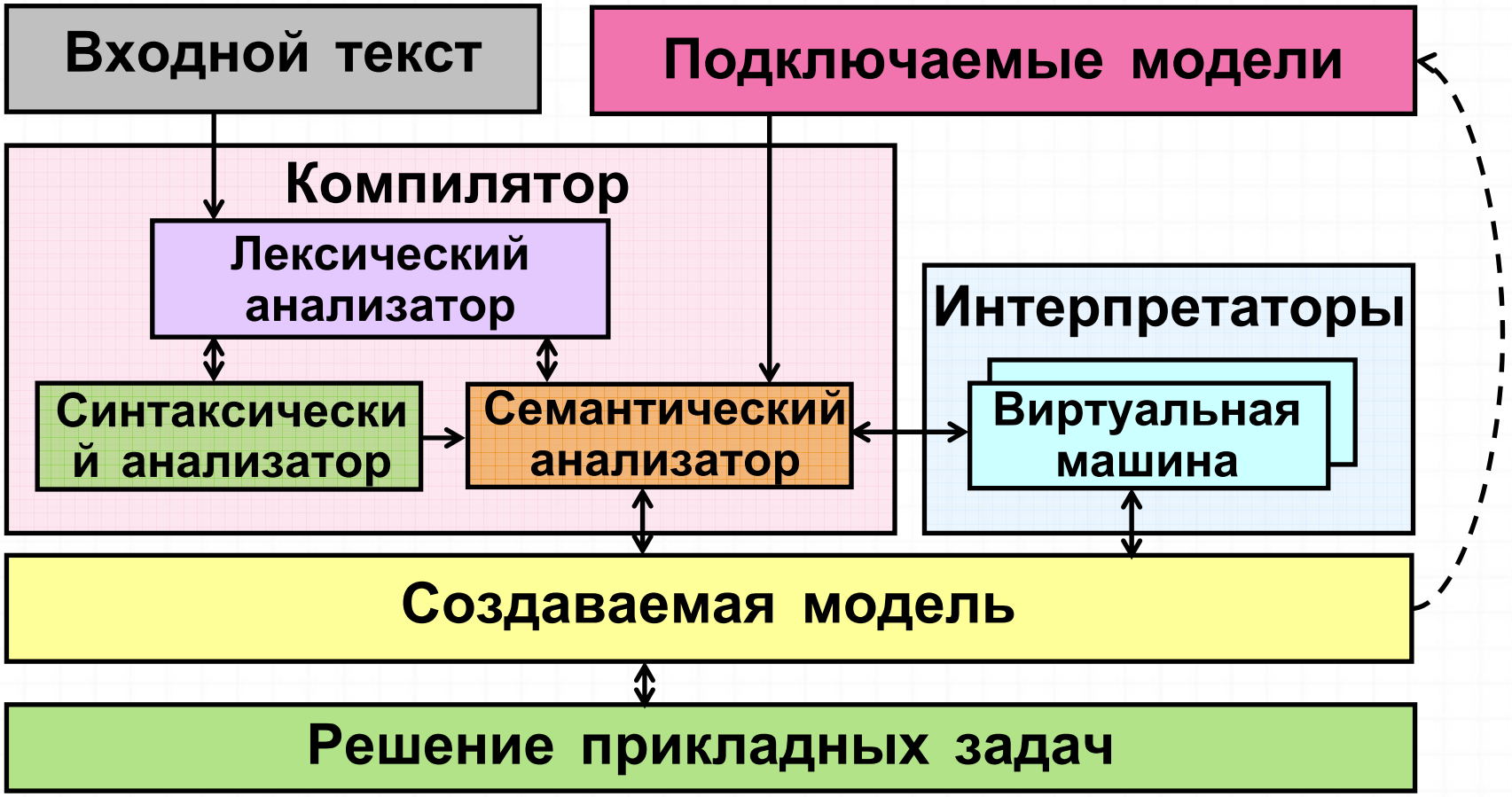
Эффективность



Анализ эффективности

A. Длина исходного описания управления лифтом, написанного экспертом на естественном языке, равна 1545 знакам, причем этот текст не содержит определение своего синтаксиса и не раскрывает семантику терминов и понятий, которые в нем использованы.
B. Длина понятийной модели и ситуационного описания равна 1606 знакам, причем понятийная модель содержит описание синтаксиса, но не содержит описание семантики.
C. Предварительная эффективность реализации системы управления лифтом – 1606/1545 = 1,04.
D. Итоговая эффективность. Если описание семантики будет реализовано с той же эффективностью, что и общее описание, то эффективность всего решения существенно не измениться.
E. Вывод: семантический разрыв между текстом исходного описания и текстом программы практически отсутствует.

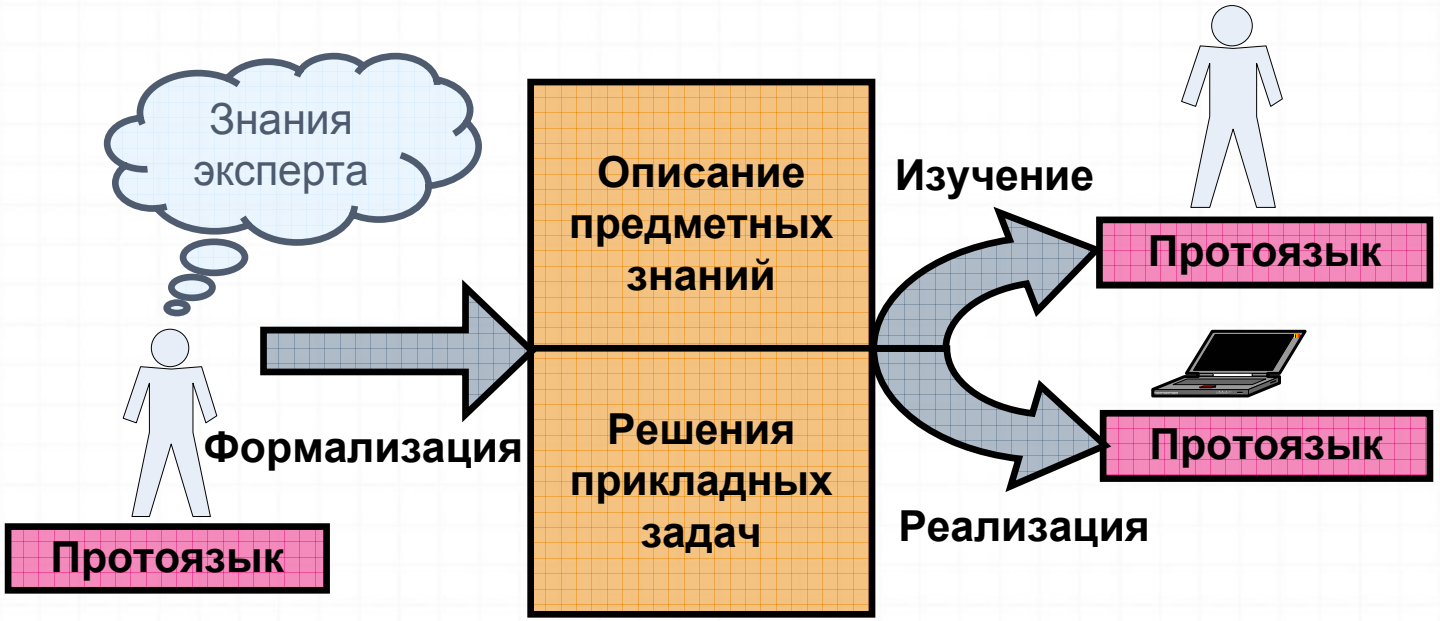
Среда программирования



Подключаемые модели:
- стандартные;
- дополняемые;
- библиотечные;
- общедоступные.

Интерпретаторы:
- контроллеры, процессоры;
- абстрактные машины;
- операционные системы;
- среды программирования.

Применение


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