Формализация интерфейсных стандартов на практике

В. Кулямин

Институт системного программирования РАН

План

- Что такое интерфейсные стандарты
  - Определения и цели
  - Типология, характеристики и примеры
  - Источники и поддержка
- Что такое формализация и что она дает
  - Цели
  - Как это выглядит: модели, процесс, примеры
  - Результаты, затраты и выгоды
- Итоги

Что такое стандарт

Набор норм и правил, регламентирующих выполнение определенных действий или характеристики процессов, продуктов и услуг в некоторой области

Стандарты ***де юре*** и ***де факто***

Поддержка стандартов

Международные организации

- ISO, Международная организация по стандартизации 1947
- IEC, Международная электротехническая комиссия 1906
- ITU, Международный телекоммуникационный союз 1865

Специализированные международные организации

- IEEE Институт инженеров по электронике и электротехнике
- ECMA ECMA International (Европейская ассоциация производителей компьютеров)
- W3C Консорциум мировой сети (включает IETF – группу развития Интернет)
- OASIS Организация по развитию стандартов на структурированную информацию
- OMG
- Open Group
- Linux Foundation (FSG + OSDL)

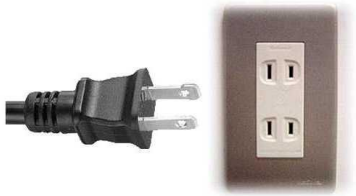
Региональные организации

- США ANSI, NIST, SEI, DoD, DoE
- Европа ETSI
- СНГ Межгос. совет по стандартизации, метрологии и сертификации (ГОСТ)

Зачем нужны стандарты

- Взаимодействие разных организаций, устройств, систем и процессов
  - **Интерфейсные стандарты**
- Предотвращение ущерба
  - Стандарты безопасности
- Удовлетворение потребителей
  - Стандарты качества

Электрические вилки и розетки I



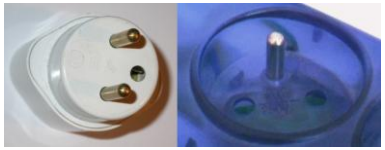
Тип A. США.

Тип B. США.

Тип C (CEE 7/16). Европа.

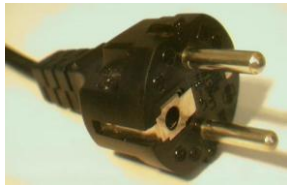
Тип C (CEE 7/17). Европа.

Тип D. Индия, Пакистан, ЮАР, Судан, Ливия, Конго.



Тип E. Франция.

Тип F (CEE 7/4). Германия.



Комбинированная вилка E+F (CEE 7/7).

Тип G. Великобритания, Кипр, Гамбия, Кения, Бахрейн и пр.

Тип H. Израиль.

Тип I. Австралия.

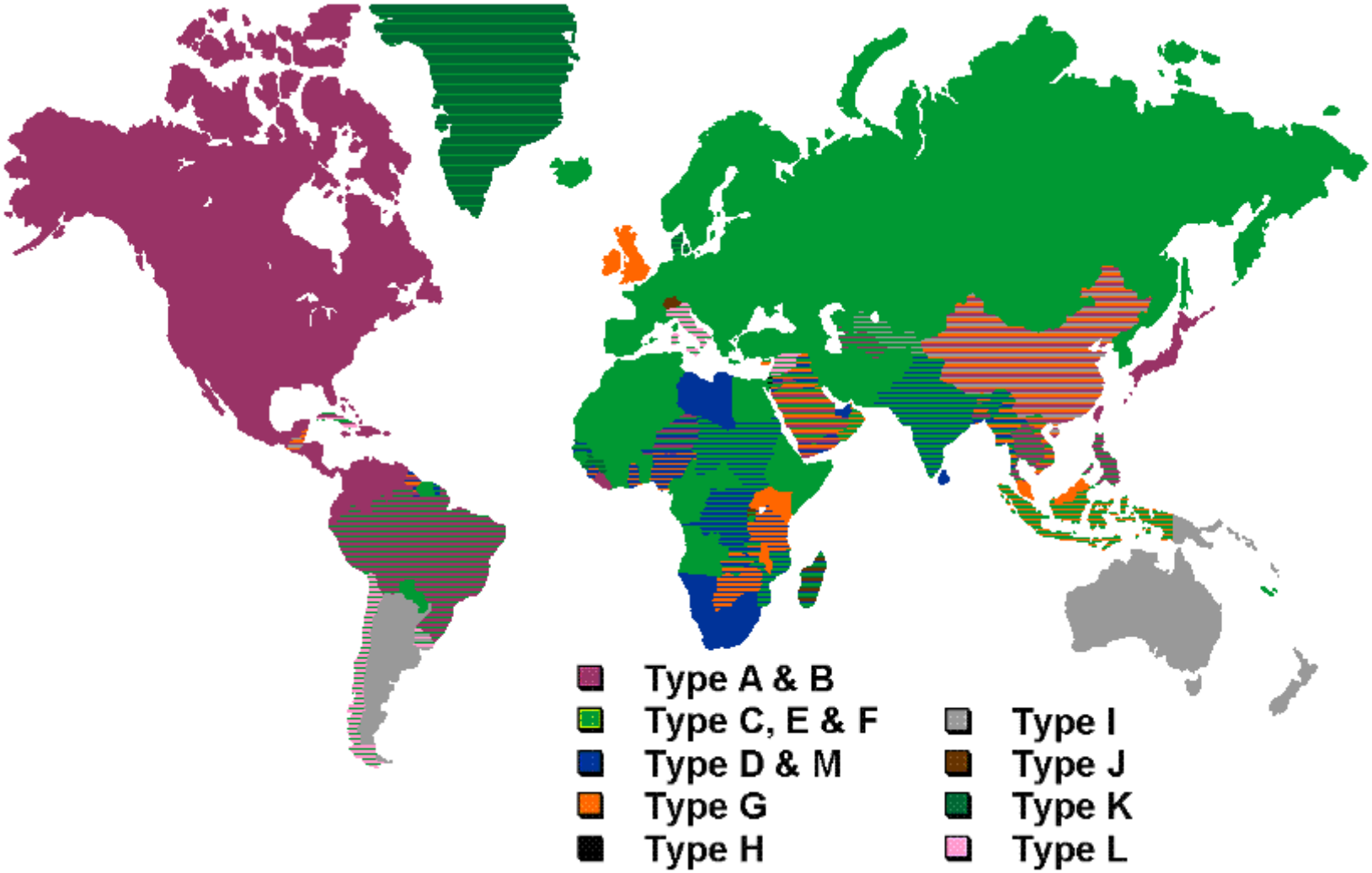
Тип J. Швейцария.

Тип K. Дания.

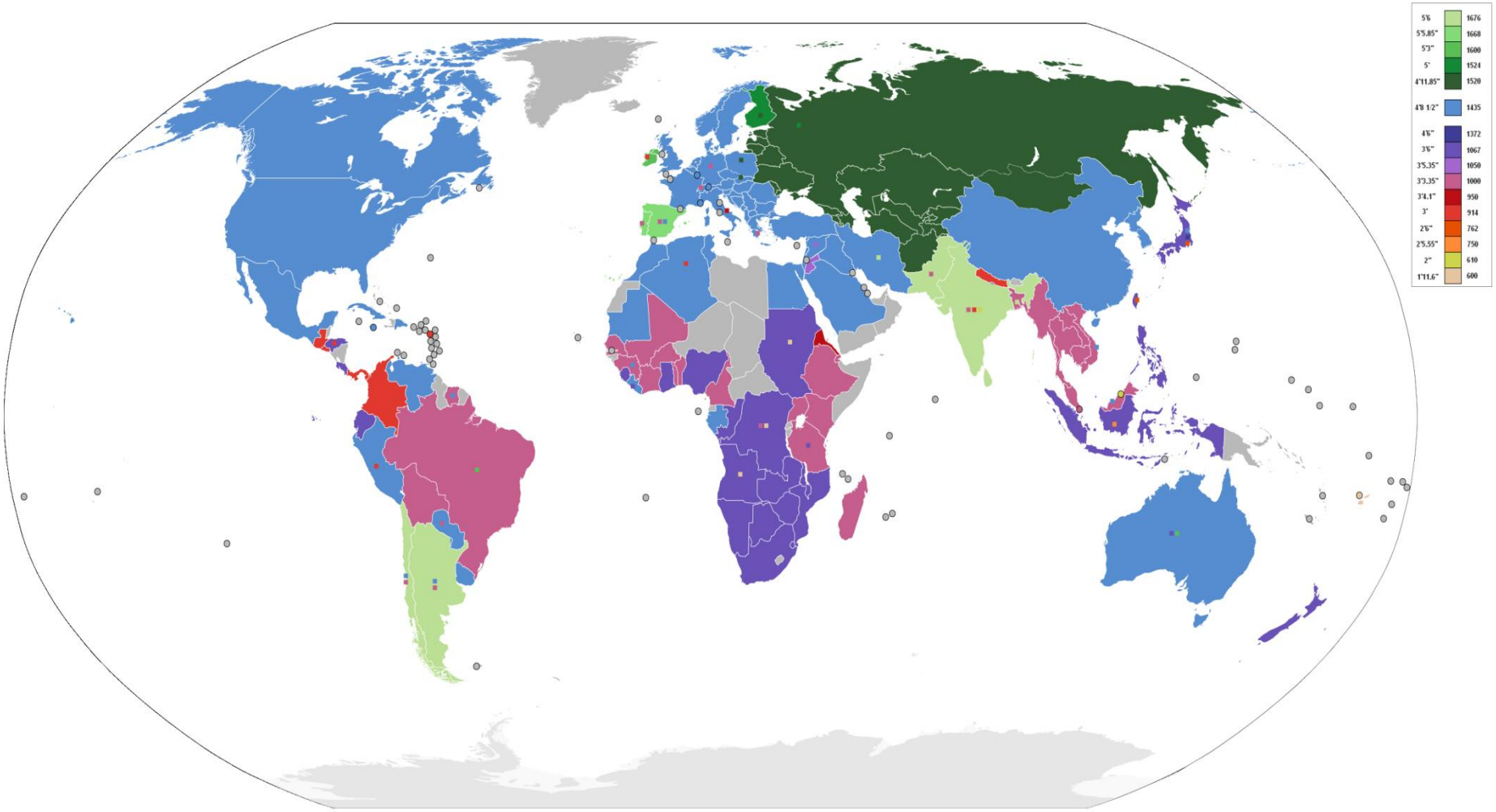
Тип L. Италия.

Тип M. Индия, Пакистан, ЮАР, Ливия, Судан, Конго.

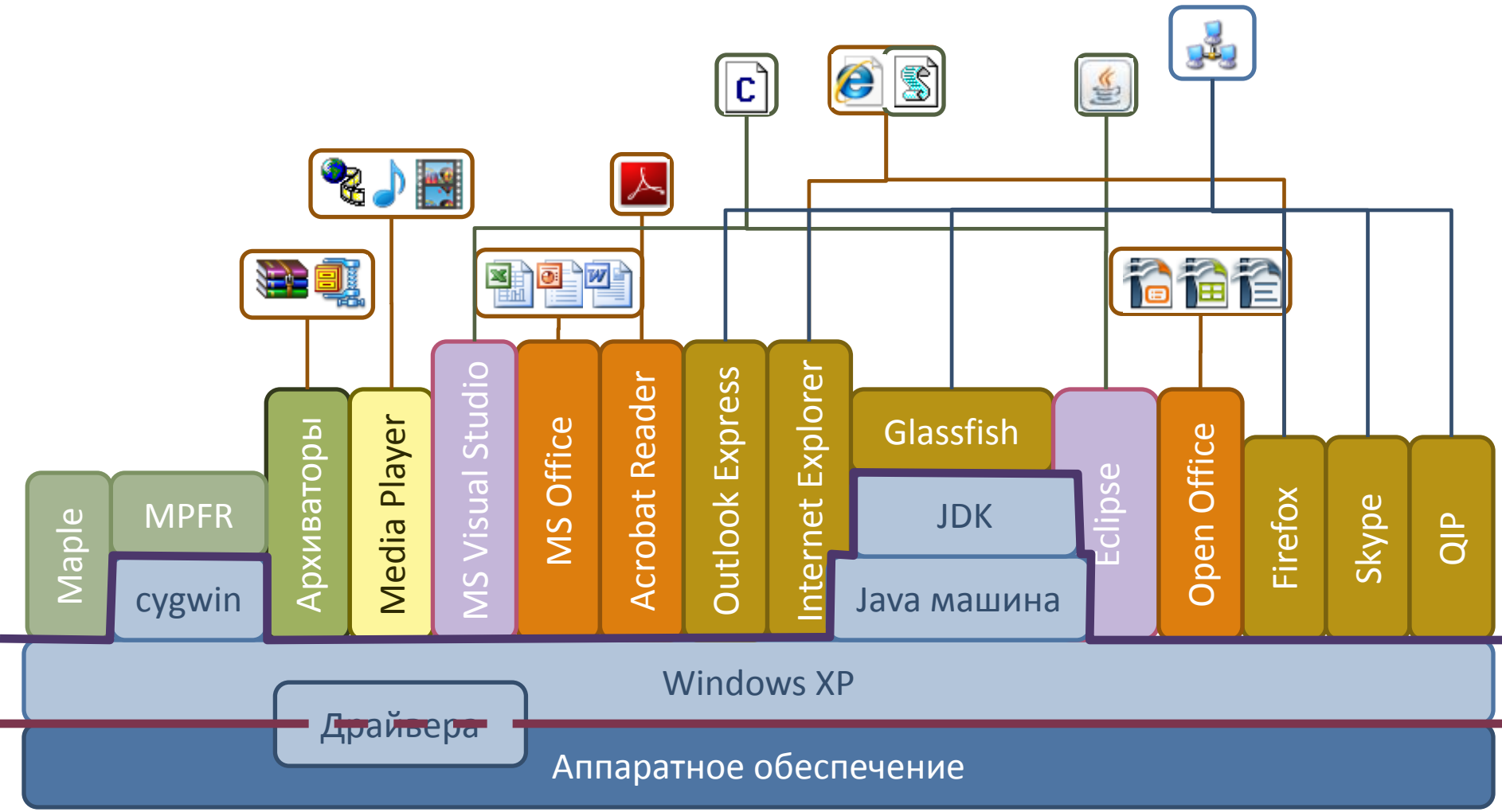
Электрические вилки и розетки II



Ширина железнодорожной колеи



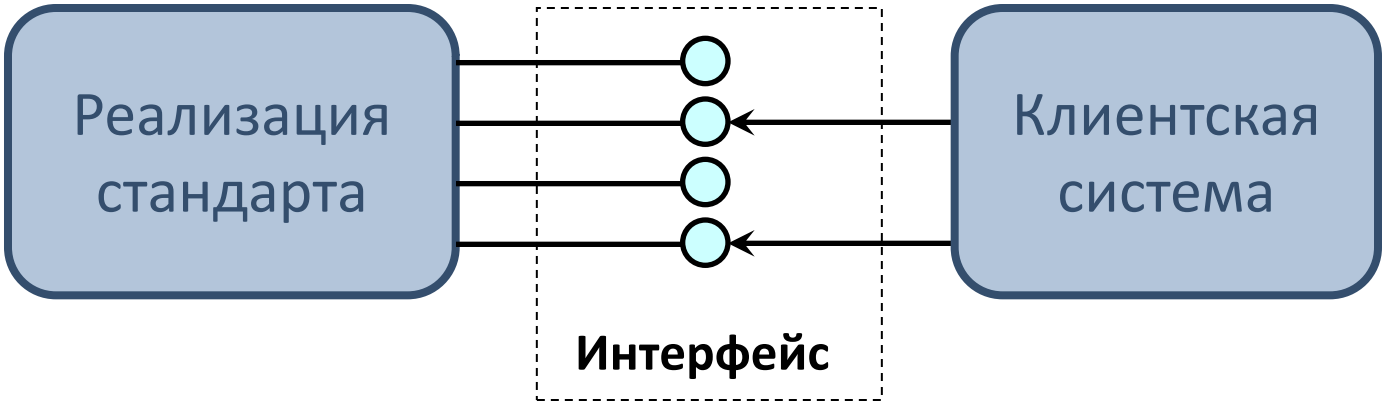
Этот ноутбук



Интерфейсные стандарты ПО

- Программно-аппаратные интерфейсы
  - x86, MIPS, ARM
- Телекоммуникационные протоколы
  - Ethernet, Wi-Fi, IP, TCP, HTTP, SMTP, SOAP, Radius
- API общих библиотек
  - POSIX, Linux Standard Base, JDK, DOM, MPI, OpenGL
- Форматы файлов
  - XML, HTML, PDF, ODF, MP3, GIF, ZIP
- Языки программирования
  - C++, Java, C#, Fortran, Ada, Scheme, Verilog

Составляющие стандарта



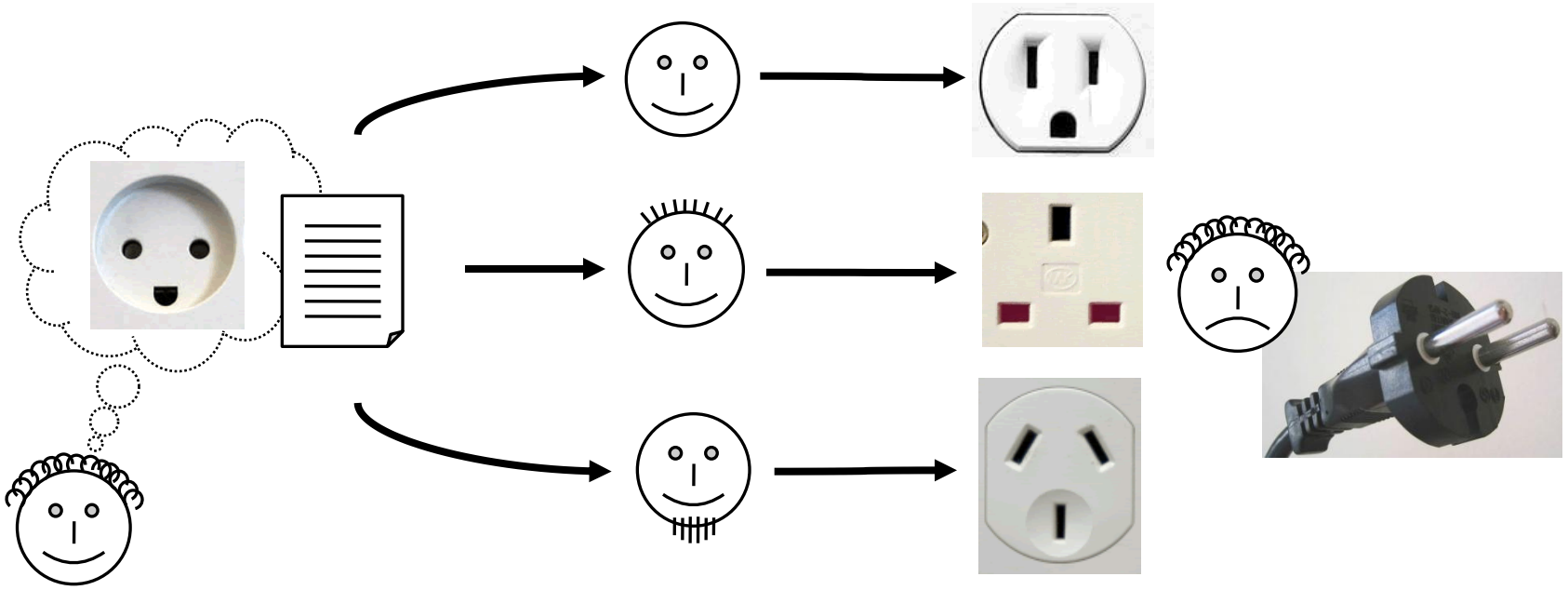
- Должны поддерживаться все указанные элементы интерфейса
  (иногда есть разбиение элементов на группы – профили)
- Каждый элемент должен соответствовать требованиям

- Доступ к необходимым ресурсам и службам должен выполняться только через описанный интерфейс
- Все задействованные элементы интерфейса должны использоваться правильно

Уровни требований

- Структурный
  - Наличие определенных элементов интерфейса
  - Сигнатуры операций и структуры данных
  - Синтаксис и статическая семантика
  - Использование только заданных элементов
  - ❖Проверяемы с помощью статического анализа
- Динамический
  - Демонстрация определенного поведения
  - Ограничения на состояния и результаты операций
  - Динамическая семантика
  - Использование в зависимости от контекста
  - ❖Проверяемы с помощью мониторинга и тестирования

Проблема однозначного понимания



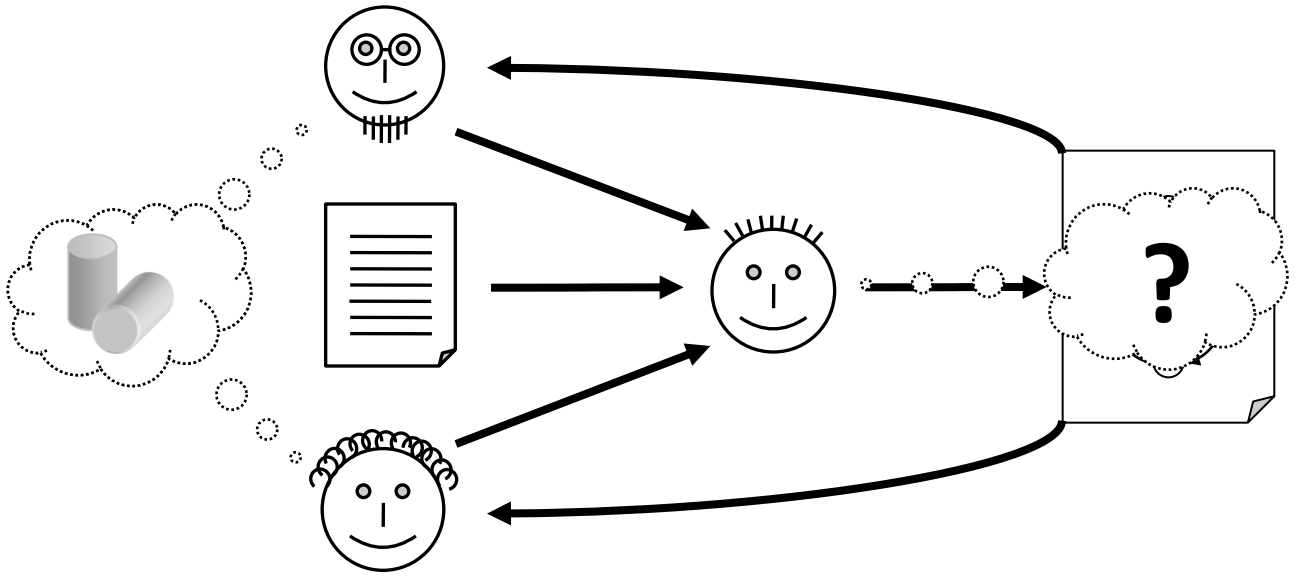
Формализация

Представление требований стандарта в виде формальной математической модели

- Зачем?
  - Возможности анализа
  - Возможности проверки (верификации)
    - Нужные свойства
      - Однозначность
      - Непротиворечивость
      - Полнота
- Адекватность – **не гарантируется!**

Как обеспечить адекватность?

- Понимание
  - Как проверить?
- Переформулировка + оценка + правка



Виды формальных моделей

- Логико-алгебраические
  - Временные логики
  - Алгебраические спецификации
  - Грамматики
  - …
- Исполнимые
  - Автоматы
  - Сети Петри
  - …
- Смешанные
  - Процессные алгебры ↔ LTS
  - ASM (машины с абстрактным состоянием)
  - Программные контракты

Опыт ИСП РАН

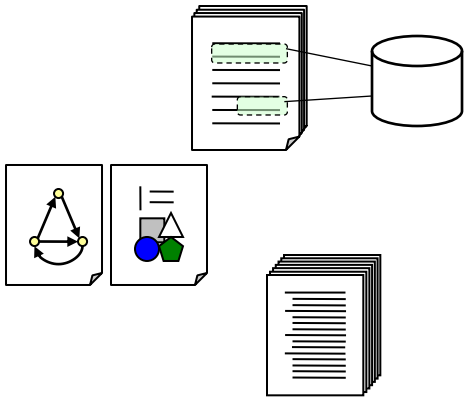
- Программно-аппаратные интерфейсы
  - MIPS
- Телекоммуникационные протоколы
  - IPv6, Mobile IPv6, IPsec, TCP, IPMP 2, SMTP, POP3
- API общих библиотек
  - JDK, POSIX $\subset$ LSB, DOM
- Форматы файлов
- Языки программирования
  - C, Java (статическая семантика)

Другие работы

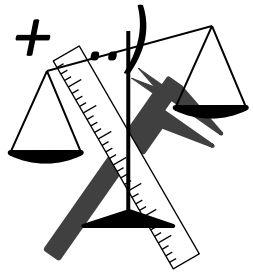
- Исследовательские работы
  - Формализация и анализ свойств протоколов
  - Формализация и анализ языков программирования
  - Формализация элементов API
- Microsoft Open Specifications

  http://www.microsoft.com/openspecifications/default.aspx

Процесс и результат формализации



- Составление каталога требований
- Концептуальное моделирование
- Формальное моделирование
- *Разработка верификационных инструментов (статический анализ + тестирование + …)*

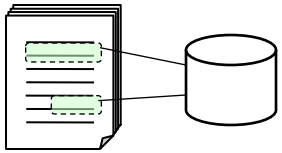


Отчуждаемый результат?

☹ Формальные модели

🙂 Инструменты проверки требований и тесты

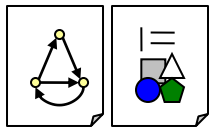
Составление каталога требований



- Определение источников
- Идентификация отдельных требований
  - Разметка документов
  - Объединение одинаковых требований из разных мест
  - Переформулировка, если надо
  - Присвоение идентификаторов
  - Установление связей с элементами исходного текста
- Классификация
  - Требования к чему (компонент/система/окружение)
  - Степень обязательности (must, shall, should/can, may)
  - Проверяемость (возможно/очень трудно/невозможно)
  - Структурные/динамические
  - Позитивные/негативные
  - Нормальная работа/обработка ошибок/серьезные сбои
- Инспекция каталога

Концептуальное моделирование

- Построение целостной модели поведения стандартизованного компонента или (под)системы



- Анализ
  - Противоречия
  - Неоднозначность, неясность
  - Неполнота
  - Соответствие исходным задачам
  - Соответствие выбранным образцам решений
  - Альтернативы
- Понимание – представить информацию в другой форме
  - Текст → таблицы, диаграммы, схемы
  - Таблицы → диаграммы, грамматики
  - Диаграммы → структурированный текст, таблицы

Пример I: `strtod()`, POSIX

`strtod` – преобразование строки в число с плавающей точкой

- Интерфейс

  double strtod(**const char ***_nptr_**, char ****endptr)
- Описание
  - Преобразует начало строки `nptr` в число типа double
  - Декомпозиция исходной строки
    - Начальные пробелы
    - Далее – то, что можно считать числом
    - Нераспознанные символы (начало записывается в `endptr`)
  - Преобразование распознанной части в число
- Результаты
  - При успешной конвертации – число
  - Иначе
    - При переполнении – +HUGE_VAL или -HUGE_VAL, errno := [ERANGE] (shall)
    - При слишком малом результате – errno := [ERANGE] (shall)
    - Иначе 0, errno := [EINVAL] (may)

Пример I: текст стандарта

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character, then an optional exponent part
- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character, then an optional binary exponent part
- One of INF or INFINITY, ignoring case
- One of NAN or NAN(*n-char-sequence*$_{opt}$), ignoring case in the NAN part, where:

n-char-sequence:
digit
nondigit
n-char-sequence digit

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

in place of a period, and that if neither an exponent part nor a radix character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it were a floating constant that is too large for the range of the return type. A character sequence NAN or NAN(*n-char-sequence*$_{opt}$) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence part that does not have the expected form; the meaning of the *n*-char sequences is implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value resulting from the conversion is correctly rounded.

[CX] The radix character is defined in the program's locale (category *LC_NUMERIC).* In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period ('.').

In other than the C [CX] or POSIX locales, other implementation-defined subject sequences may be accepted.

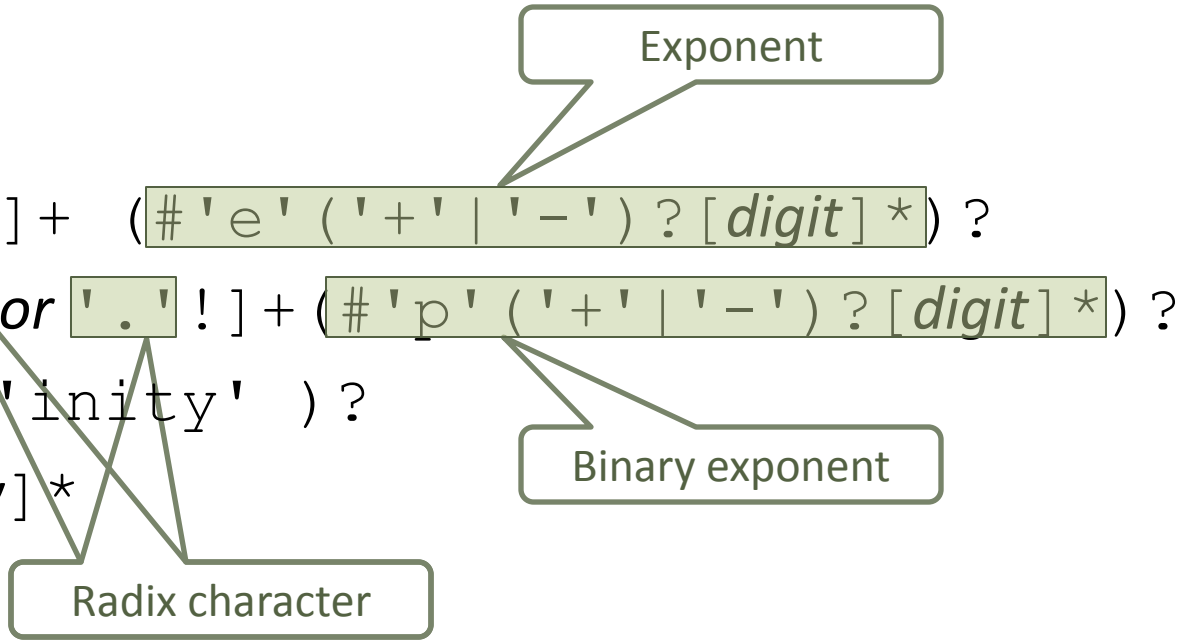
If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

[CX] The *strtod*() function shall not change the setting of *errno* if successful.

Since 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *strtod*(), *strtof*(), or *strtold*(), then check *errno*.

Пример I: разбор строки

```
(  [whitespace]  )*
(  '+'  |  '-'  )?
(
      [digit or '.'!]+  (#'e'('+'|'-')?[digit]*)?
   |  #'0x'[hdigit or '.'!]+(#'p'('+'|'-')?[digit]*)?
   |  #'inf'  (  #'inity'  )?
   |  #'nan'  [any]*
)
```



Пример I: результаты `strtod()`

- не число → `result = 0,`
  `endptr = &nptr`
  `errno =`$_{may}$ `EINVAL`
- overflow → `result = ±HUGE_VAL,`
  `endptr = …,`
  `errno = ERANGE`
- underflow → `result = $(…),`
  `endptr = …,`
  `errno = ERANGE`
- normal → `result = …,`
  `endptr = …,`
  `errno не менять`

Похожий пример из Qt

double QString::toDouble (**bool** * ok = 0) **const**

Returns the string converted to a double value.
Returns 0.0 if the conversion fails.
If a conversion error occurs, **ok* is set to false; otherwise **ok* is set to true.

Various string formats for floating point numbers can be converted to double values.

This function tries to interpret the string according to the current locale. The current locale is determined from the system at application startup and can be changed by calling QLocale::setDefault(). If the string cannot be interpreted according to the current locale, this function falls back on the "C" locale.
Due to the ambiguity between the decimal point and thousands group separator in various locales, this function does not handle thousands group separators. If you need to convert such numbers, see QLocale::toDouble().
See also number(), QLocale::setDefault(), QLocale::toDouble(), and trimmed().

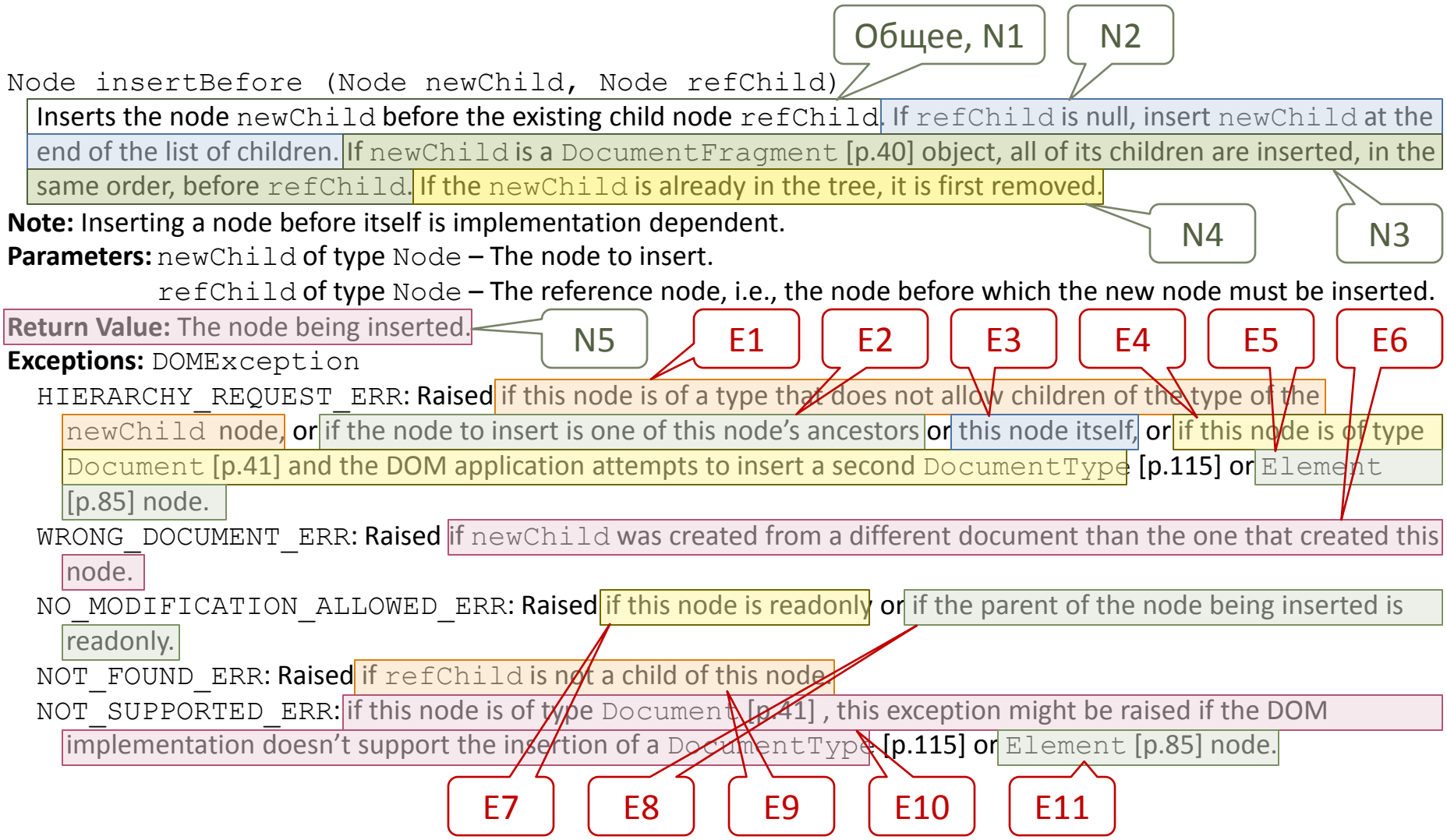
Формальное моделирование

- Программные контракты (для API, протоколов)
  - Предусловия операций
  - Постусловия операций
  - Модель состояния
  - Инварианты типов данных
  - Асинхронные действия и семантика чередования
- Порождающие грамматики (для языков)
- Прослеживаемость требований
- Автоматизированное построение тестов
  - Абстрактная автоматная модель
  - Автоматический обход автомата + генераторы данных
  - Привязка к тестируемой системе через адаптеры

Пример I: спецификация

```
specification Unifloat* strtod_spec(CString* st, CString** endptr, ErrorCode* errno) {
  pre { REQ("", "endpt should be not NULL", endptr != NULL); return true; }
  post {
    CString* model_endptr; IntT model_err = 0;
    Unifloat* model_res = strtod_model(st, &model_endptr, &model_err);
    round_Unifloat(strtod_spec); round_Unifloat(model_res);
    if (model_err == 1) {
      REQ("strtod.13", "If no conversion is performed, the value of nptr shall be stored"
                       "in the object pointed to by endptr", equals(st, *endptr)); }
    if (*errno == SUT_EINVAL) {
      REQ("strtod.15", "If no conversion could be performed, 0 shall be returned"
          , isZero_Unifloat(strtod_spec)); }
    if (isOverflow_Unifloat(model_res)) {
      REQ("strtod.16", "If the correct value is outside the range of representable values,"
                       "HUGE_VAL shall be returned", isInfinity_Unifloat(strtod_spec)); }
    if (isUnderflow_Unifloat(model_res)) {
      REQ("strtod.17", "If the correct value would cause underflow, the smallest normalized"
                       "positive number shall be returned"
          ,    (compare_Unifloat(abs_Unifloat(strtod_spec), min_Unifloat(type)) != 1)
            && (strtod_spec->sign == 1)); }
    ...
```

Пример II: `insertBefore()`, DOM

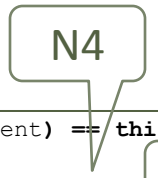


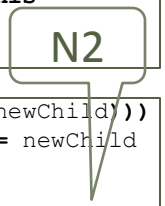
Пример II: спецификация исключений

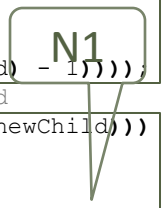
```
Node Node.InsertBefore(Node newChild, Node refChild)
{
  Contract.Requires(newChild != null);    // What if newChild is null? DOM specification says nothing
  Contract.EnsuresOnThrow<DomException> (
  // If this node is of a type that does not allow children of the type of the newChild node
       !IsAllowedChild(newChild) && !(newChild is DocumentFragment)
  // -- in particular, if inserted DocumentFragment has a prohibited child for this node
    ||    newChild is DocumentFragment && newChild.Children.Count > 0
       && Contract.Exists(0, newChild.Children.Count, i => !IsAllowedChild(newChild.Children[i]))
  // or this node is of type Document and newChild is a second DocumentType or Element node
    || this is Document && newChild is DocumentType && HasDifferentChildOfType(typeof(DocumentType), newChild)
    || this is Document && newChild is Element && HasDifferentChildOfType(typeof(Element), newChild)
  // -- in part. if inserted DocumentFragment has a DocumetType or an Element child, and this node has one too
    ||    this is Document && newChild is DocumentFragment && HasDifferentChildOfType(typeof(DocumentType), null)
       && newChild.HasDifferentChildOfType(typeof(DocumentType), null)
    ||    this is Document && newChild is DocumentFragment && HasDifferentChildOfType(typeof(Element), null)
       && newChild.HasDifferentChildOfType(typeof(Element), null)
  // if newChild is this node itself or if newChild is one of this node's ancestors
    || newChild == this || Ancestors.Contains(newChild)
  // if newChild was created from a different document than the one that created this node
    || !(this is Document) && OwnerDocument != newChild.OwnerDocument
  // -- (refinement) owner document for Document is null, whether newChild owner is other document
    || this is Document && this != newChild.OwnerDocument
  // if this node is readonly or if the parent of the node being inserted is readonly
    || IsReadOnly || newChild.Parent != null && newChild.Parent.IsReadOnly
  // if refChild is not a child of this node
    || refChild != null && !Children.Contains(refChild) );
```

E1 E2 E3 E4 E5 E6 E7 E8 E9

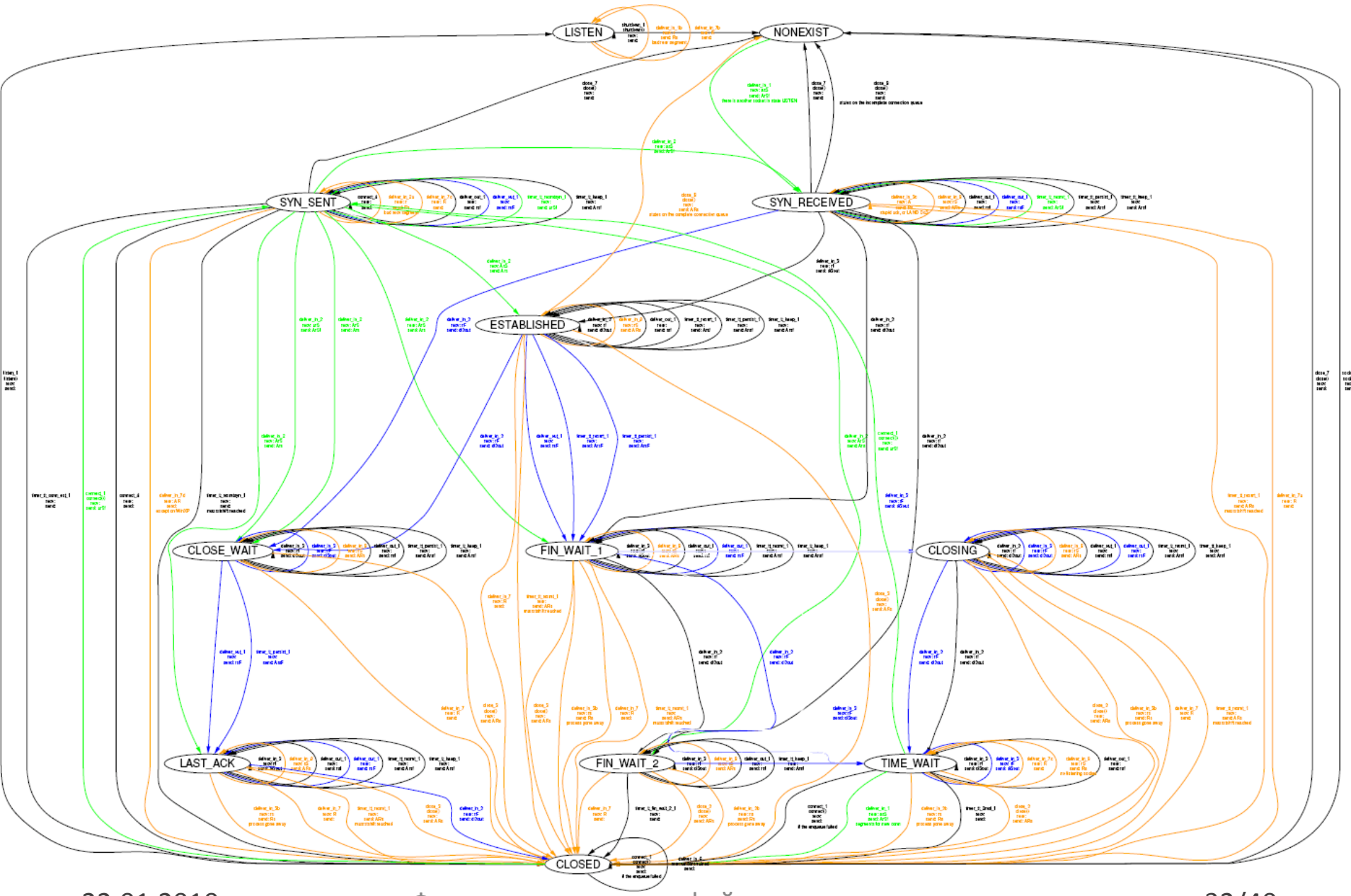
Пример II: спецификация нормы








```
// If the newChild is already in the tree, it is first removed
Contract.Ensures(    Contract.OldValue<Node>(newChild.Parent) == null || Contract.OldValue<Node>(newChild.Parent) == this
                 || !Contract.OldValue<Node>(newChild.Parent).Children.Contains(newChild) );
Contract.Ensures( !(newChild is DocumentFragment) || newChild.Children.Count == 0 );
// If refChild is null and newChild is not DocumentFragment, insert newChild at the end of the list of children
Contract.Ensures( !(refChild == null && !(newChild is DocumentFragment) && !Contract.OldValue<bool>(Children.Contains(newChild)))
                 ||    Children.Count == Contract.OldValue<int>(Children.Count) + 1 && Children[Children.Count - 1] == newChild
                    && Children.GetRange(0, Children.Count - 1).Equals(
                         Contract.OldValue<List<Node>>(Children.GetRange(0, Children.Count))) );
Contract.Ensures( !(refChild == null && !(newChild is DocumentFragment) && Contract.OldValue<bool>(Children.Contains(newChild)))
                 ||    Children.Count == Contract.OldValue<int>(Children.Count) && Children[Children.Count - 1] == newChild
                    && Children.GetRange(0, Contract.OldValue<int>(Children.IndexOf(newChild))).Equals(
                         Contract.OldValue<List<Node>>(Children.GetRange(0, Children.IndexOf(newChild))))
                    && Children.GetRange(  Contract.OldValue<int>(Children.IndexOf(newChild))
                                         , Children.Count - Contract.OldValue<int>(Children.IndexOf(newChild)) - 1)
                       .Equals(Contract.OldValue<List<Node>>(
                          Children.GetRange(Children.IndexOf(newChild) + 1, Children.Count - Children.IndexOf(newChild) - 1))));
// If refChild isn't null and newChild is not DocumentFragment, insert newChild before the existing child node refChild
Contract.Ensures( !(refChild != null && !(newChild is DocumentFragment) && !Contract.OldValue<bool>(Children.Contains(newChild)))
                 ||    Children.Count == Contract.OldValue<int>(Children.Count) + 1
                    && Children.IndexOf(newChild) == Contract.OldValue<int>(Children.IndexOf(refChild))
                    && Children.GetRange(0, Children.IndexOf(newChild)).Equals(
                         Contract.OldValue<List<Node>>(Children.GetRange(0, Children.IndexOf(refChild))))
                    && Children.GetRange(Children.IndexOf(newChild) + 1, Children.Count - Children.IndexOf(newChild) - 1)
                       .Equals(Contract.OldValue<List<Node>>(
                          Children.GetRange(Children.IndexOf(refChild), Children.Count - Children.IndexOf(refChild)))) );
Contract.Ensures( !(    refChild != null && !(newChild is DocumentFragment)
                    && Contract.OldValue<bool>(Children.Contains(newChild)) && newChild == refChild )
                 ||    Children.Count == Contract.OldValue<int>(Children.Count)
                    && Children.IndexOf(newChild) == Contract.OldValue<int>(Children.IndexOf(refChild))
                    && Children.GetRange(0, Children.Count).Equals(
                         Contract.OldValue<List<Node>>(Children.GetRange(0, Children.Count))) );
...
```


LISTEN
NONEXIST
SYN_SENT
SYN_RECEIVED
ESTABLISHED
CLOSE_WAIT
FIN_WAIT_1
CLOSING
LAST_ACK
FIN_WAIT_2
TIME_WAIT
CLOSED

Пример IV: мат. функции POSIX

NAME sin, sinf, sinl - sine function

SYNOPSIS #include <math.h> double sin(double *x*); float sinf(float *x*); long double sinl(long double *x*);

DESCRIPTION

These functions shall compute the sine of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the sine of *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

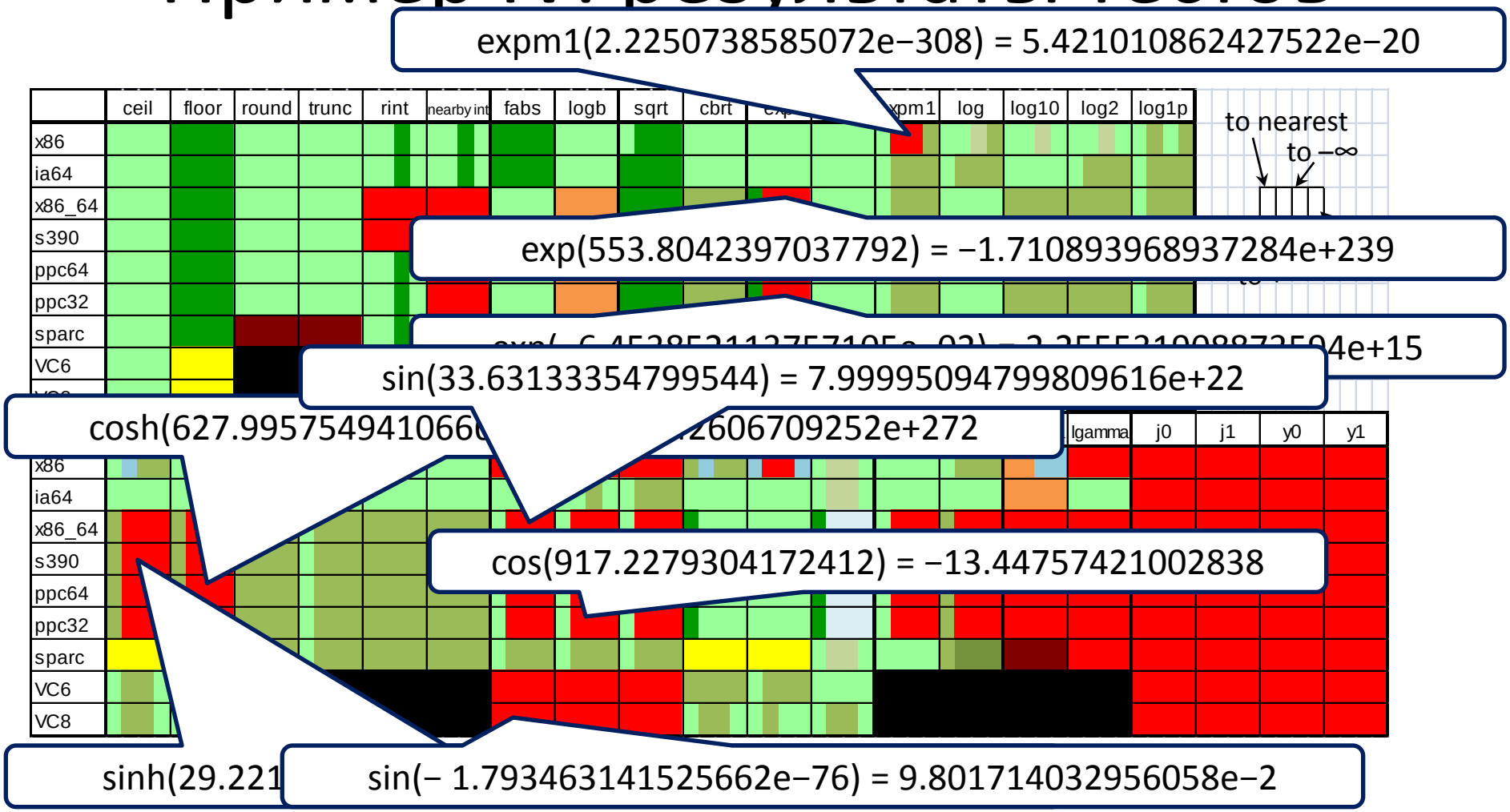
These functions shall fail if:

Domain Error The *x* argument is ±Inf. If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

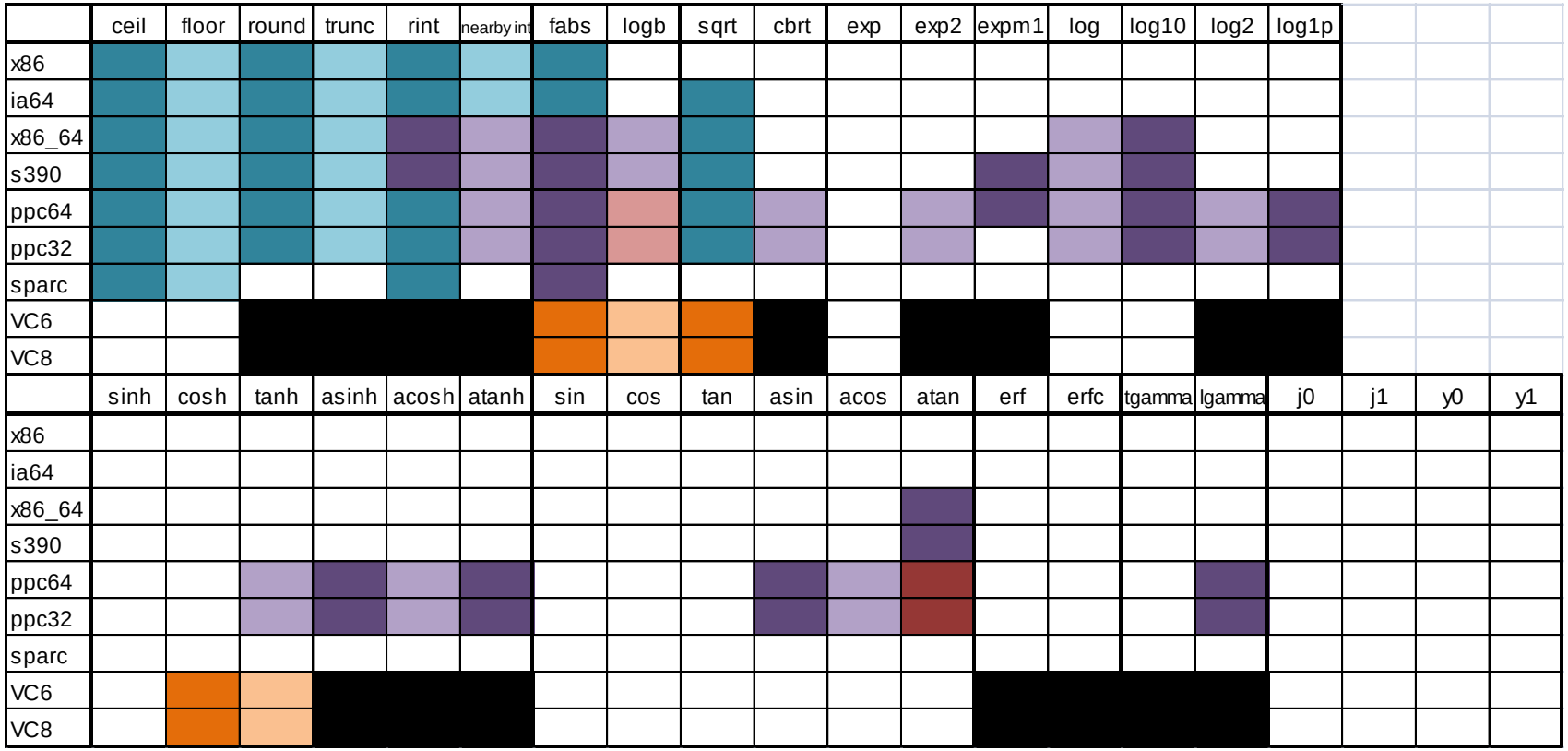
These functions may fail if:

Range Error The value of *x* is subnormal If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

Пример IV: результаты тестов



Пример IV: различия результатов



Unsupported

Строгость стандартов

- Программно-аппаратные интерфейсы
  - Высокая
- Телекоммуникационные протоколы
  - Высокая для простых протоколов, понижается при росте сложности
- API общих библиотек
  - Средняя для широко используемых и давно известных библиотек, низкая для новых
- Форматы файлов
  - ???
- Языки программирования
  - Средняя, выше для более простых и новых «серьезных» языков

Что дает формализация

- Приближение к совместимости
  - Выявление неоднозначности
  - Выявление рассогласований, альтернативных решений
  - Выявление неполноты
- Возможность строгой проверки соответствия при разработке
  - Правила статического анализа и верификации
  - Тесты

Стоимость

- 2-3 человеко*дня за элемент в среднем
- Вариации по сложности
  - 7-10 ч*д для сложных элементов
  - 0.5-2 ч*д для средних
  - 0.05-0.1 ч*д для простейших

Итоги

- Интерфейсные стандарты нуждаются в однозначности, непротиворечивости и полноте
- Формализация – техника достижения этого
- Полная формализация довольно дорога, зато повторяема
- Частичная формализация доступнее по цене, но искусство
- Формализация не дает понимания сама по себе, оно достигается за счет переформулировки
- Тесты и инструменты статического анализа – основной отчуждаемый результат
- Стандарт должен стабилизироваться
- Стандарт хорошо поддерживать на базе некоторой информационной системы

Спасибо за внимание!

Вопросы?

В. В. Кулямин kuliamin@ispras.ru

www.ispras.ru/~kuliamin

Отдел технологий программирования ИСП РАН

www.ispras.ru/ru/se/index.php

www.unitesk.ru

www.linuxtesting.ru

зав. А. К. Петренко petrenko@ispras.ru