synrc research center s.r.o.
Roháčova 141/18, Praha 3 13000, Czech Republic

Системна інженерія та верифікація уніфікованих обчислювальних середовищ

System Engineering and Verification of Unikernel Executional Environments

Максим Сохацький

Київський Політехнічний Інститут
Листопад 2015

1 Вступ

1.1 Що означає назва

«Формальні методи верифікованих середовищ послідовних обчислень розподілених узгоджених паралельних систем»

Формальні методи тут розглядаютьяся у широкому сенсі, особливо нас цікавитиме та частина математичного забезпечення яку можна формалізувати для доведення машинним способоб, що унеможливлює виникнинення цілого класу помилок. Зокрема розглядатимуться усі теорії які застосовують до подібних систем з підвищеними запитами якості.

Розглядаючи такі системи, які піддаються формальній верифікації, кажуть про сертифіковане або верифіковане програмне забезпечення. А у випадку коли такі системи розповсюджуються на всі прошарки моделі OSI, то кажуть про замкнені верифіковані середовища. Саме розробка такого середовища є основним завданням даної роботи.

Чому саме послідовні обчислення? У цій частині мова йде про природу обчислень, чи природу послідовності висловлювань у формальній логіці. Саме лінійність кінцевого процессу виконання певного структурованого або циклічного алгоритму відіграє ключову роль у моделюванні віртуальних обчислювальних середовищ. Такі лінеаризовані системи уже показали свою ефективність в певному класі обчислень, таких як ланцюгова реплікація, реактивні системи, та інші моделі напівгруп навколо певних типів – протоколів взаємодії між процесами. Крім того такі послідовності подій піддаються статистичній обробці для визначення первних кластеризацій та інших кореляцій у просторі та часі, тому можна говорити про певні зручні, нормалізовані системи типів для такого роду маніпуляцій над даними.

Побудова розподілених та паралельних, тобто здатних виконуватися на багатьох машинах одночасно, та узгоджених, тобто не блокуючих, а значить лінеаризованих систем управліннями процессами є побічним результатом який очікується від цієї роботи.

1.2 Контекст та сфера дослідження

За багато років кількість теорій, які використовуються для побудови програмного забезпечення значно розширилися: починаючи з теорії компіляції сучаних функціональних мов, та систем програмування на основі теорії типів, включаючи сучасні моделі обчислень, які побудовані на основі лямбда числення та числення процесів, закінчуючи віртуальними машинами які працюють у семантиці захищених, простих за структурою процесів, час яких розподіляється у прозорий та ефективний спосіб.

Сучасний розвиток техніки та теоретична межа швидкості обробки процесорів вивів на передній план алгоритми та структури данних які ефективно використоують розподілені у просторі та часі ресурси, як то об'єми памяті та обчислювальні потужності. Принципи та підходи паралельного та узгодженого програмування дають змогу масштабувати системи та обчисленя, однак анонсують нові теорії для забезпечення коректності в умавах підвищеної складності алгоритмів у розподілених системах, таких як алгоритми забезпечення консистентності та транзакційності у розподілених системах PAXOS та CR.

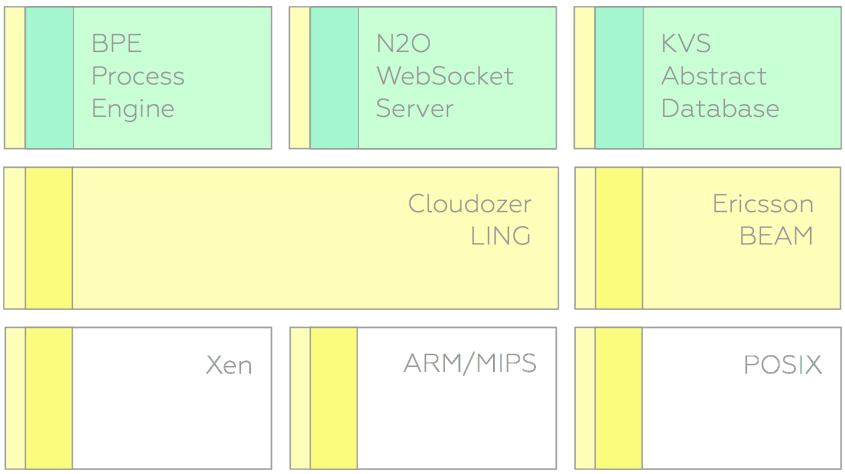
З розвитком систем програмування, підвищилась якість інструментів для автоматичного доведення коректності, починаючи від інструментів побудованих на темпоральній логіці як TLA+ та NuPRL закінчуючи системами побудованими на індуктивних типах і які вже використовуються метематиками як Agda та Coq.

Багато методологій виникло і багато підходів були випробувані в польових умовах на різних, по критичності, рівнях якості. Починаючи від асемблерів, через процедурний та об'єктно-орієнтований підхід, до функціональних мов програмування з розвиненими системами типів, таких як Haskell та ML, а також мов, спеціалізованих для розподілених систем як Erlang.

Застосовуючи формальні методи доведення коректності велику частину має повнота за макненість системи, адже недоведені або неверифіковані частини можуть вплинути на детермінованість а отже якість системи. Тому має велике значення забезпечення виконання системи якогомога ближче до аппаратного забезпечення. Таких напрямків побудови замкнених середовищ є всього три, на мовах Erlang, Haskell та OCaml.

1.3 Завдання дослідження

Тактична мета даної роботи – це реалізація усіх верхніх компонентів OSI на базі метамоделі Exe на мові Erlang. Побудова зручної сучасної гнучкої верифікованої теорії з компактною системою типів є основним завданням даного дослідження.



Верифікована система типів попереднього рівеня моделі OSI — віртуальної машини є предметом майбутніх досліджень. Адам Чіпала [25] показав як можно верифікувати виконання команд віртуальної машини та компіляцію цих команд в байт-код.

Сховище даних. У цій роботі ми будемо верифікувати системні бібліотеки для майбутньої верифікованої віртуальної машини. Зокрема значна увага буде приділятися послідовностям. Система персистентного зберігання послідовностей є ядром нашого обчислювального середовища. Властивості цієї підсистеми є основоположними для перевірки констистентності операційних логів розподілених базах данних.

Бізнес-процеси. Тут ми дамо формальне визначення та верифікуємо типи системи управління структурованими процесами або FSM автоматами. Бізнес-процеси це зручний та ефективний спосіб упорядкування та формалізації потоків даних. Ми зосередимося на найкомактніших алгебрах.

Сервер додатків. Ця частина дасть відповіть на питання структурування протоколів та визначення універсального мультиплікатора. Буде описаний повний стек протоколів для IoT та WebSocket додатків.

1.4 Об'єкт дослідження

Об'єктом дослідження є усі можливі моделі обчислювальних середовищ в основному придатних до верифікованого аналізу та обробки формальними методами. Самі формальні методи теж є частиною об'єкта дослідження. Ми досліджуємо ті структури та алгоритми які дадуть максимально ефективний спосіб кодування та виконнання, забезпечуючи при цьому семантику, яка використовуються для машинного доведення коректності роботи алгоритмів та непротиречивості структур даних.

Формальні мови програмування такі як Haskell та OCaml є надзвичайно потужними з точки зору теорії типів, що дало би змогу більш формально на ранніх етапах застосувати типізацію. З іншого боку Erlang пропонує більш довершені та прості маханізми забезпечення віртуального середовища як на голому залізі так і під усправлінням гіпервізора Xen. Так як нам усе одно доведеться конвертувати специфікації з розробленої нами формальної теорії Exe, тому вибір мови з розвиненою системою типів не є основним критерієм. Для побудови замкненої системи була вибрана мова Erlang та її віртуальні машини LING та BEAM.

Усі моделі які побудовані в рамках запропонованої методології із застосуванням метамови та метамоделі операційного середовища Exe виконуватимуться у контексті віртуальних машин на голому залізі без застосування операційної системи у ході виконання. Такі системи називаються системами з уніфікованим ядром яке пропонує свою модель виконання, у нашому випадку це віртуальні машини Erlang та комплекс програмного забезпечення розроблений для замкненого циклу виробництва.

1.5 Теоретичні сутності

У таксономії теоритичних сутностей умовно визначатимемо графічні топологічно-структурні, аналітичні, статистичні та логічні типи теорій, які ми використовуватимемо для опису комплесної теорії верифікованих середовищ послідовних обчислень розподілених узгоджених паралельних систем.

Теорія масового обслуговування

Теорія массового обслуговування застосовується для побудови статистичних моделей та запобігання відмов. Перші роботи у цій області належать шведському математику Агнеру Крурупу Ерлангу, який займався дослідженнями трафіка у телефонних мережах. Модель масового обслуговування достатньо адекватно описує роботу віртуальної машини Erlang, де клієнти – це процеси, які мають черги повідомлень, та здатні відправляти заявки на обслуговування у такі самі черги інших процесів. Ці заявки, чи повідомлення складають певний протокол взаємодії у системі таких процесів. Тому тут теорія масового обслуговування застосовується для визначення пропускної здатності системи.

Пі-числення

Теорія Пі-числення Роберта Мілнера є основним формалізмом обчислювальної теорії та її імплементації. З часів виникнення CSP числення розробленого Хоаром, Мілнеру вдалося значно розширити та адаптувати теорію до сучасних телекомунікаційних вимог, як наприклад хендовери в мобільних мережах. Основні теорми в моделі Пі-числення стосуються непротиречивості та неблокованості у синхронному виконанні мобільних процесів. Так як сучасний Web можно розглядати як телекомунікаційну систему, тому у розробці додатків можна покладатися у тому числі і на такі моделі як Пі-числення.

Мережі Петрі

Мережі Петрі в даній роботі використовуються як прототип графічного лінгвістичного засобу структури категорій та системи типів. Оскільки такі графічні засоби як UML та різноманітні окремі технологічні стандарти як то BPMN більше допомагають ніж мішають, була розроблена також і графічна мова на базі графічної мови мереж Петрі, оскільки їх семантика ділить один простір з тематикою даної роботи. Ми візьмемо лінгвістичне забезпечення Мереж Петрі як прототип для нашої власної мови візуалізації структур обчислень.

Теорія категорій

Теорія категорій як робоча структурна теорія системи типів мови Exe та категоріальна семантика числення процесів. Можна було би використовувати абстракну алгебру загалом, та теорія напівгруп, а саме напівгруп активностей, проте ми будемо використовувати категоріальну семантику, що стало можливо завдяки роботам Лавіра та іншим.

Лямбда числення

Лямбда-числення як основна абстракція обчислювальної віртуальної машини. Будучи внутрішньою мовою декартово-замкненої категорії лямбда числення окрім змінних та констант у вигляді термів пропонує операції абстракції та аплікації, що визначає достатньо лаконічну та потужну структуру обчислень з функціями вищих порядків, та метатипизаціями, такими як System F, яка була запропонована вперше Робіном Мілнером в мові ML, та зараз присутня в більш складних, таких як System Fω системах Haskell та Scala. Наша результуюча система типів буде тегована система типів мови Erlang.

Темпоральна логіка

Темпоральна логіка як індуктивна теорія верифікації розподілених алгоритмів застосовується до доведення коррекності усіх нормалізованих підсистем. На основі теорії Леслі Лампорта [7], за яку він отримав премію Тюрінга. Елементи темпоральної логіки нам знадобляться для розробки теорій розподілених у просторі та часі алгоритмів.

Інтуітіоністична теорія типів Мартіна Льофа

Системи з залежними типами як верифікаційні математичні формальні моделі для доведення корректності. Система Σ та Π типів, як кванторів існування та узагальнення. Системами Mizar, Coq, Agda, Lean. Ми будемо використовувати саме

1.6 Предмет дослідження

Предметом дослідження даної роботи є розробка формальних методів для побудови операційних середовищ та метамоделей для їх формальної специфікації. Розглянувши усі можливі математичні моделі опису формалізму процесів ми формуємо ряд вимог корисних для побудови ефективної моделі яка дозволить:

- Скоротити об'єм коду на порядок
- Нормалізувати дані для їх статистичної обробки
- Легко обчислювати показники системи масового обслуговування
- Легко доводити коректність
- Мінімізувати цикл розробки програмного забезпечення
- Підвищити ефективність виконання

Для забезпечення стратегії цієї роботи ми визначаємо найбільш оптимальну модель потоку даних та функції для їх обробки, використовуючи алгебраїчний підхід для опису протоколів та їх категорій.

- Дослідження алгебраїчних структур середовища
- Визначення характеристик нормалізованих та оптимальних структур
- Побудова системи типів в автоматизованій системі доведення теорем
- Розробка системи додатків на основі метамоделі на мові Erlang
- Впровадження, діагностика та аналіз роботи системи

Іншими словами словом ми розроблюємо теорію та імплементацію мінімалістичного сертифікованого верифікованого операційного середовища для компонентів замкненої системи: абстракції аппаратного забезпечення, мова програмування, віртуальна машина, комунікаційні черги, бази даних, веб сервери, сервери додатків, та усе, що піддається верифікації та по можливості є коректним за побудовою. Починаючи з фундаментального формалізму лямбда та пі числення, через абстракції віртуальної машини до віртуальної апаратури, реальні сучасні додатки та протоколи, закінчуючи прикладами, засобами та документацією на створене обчислювальне середовище та його модель.

Алгебраїчний підхід

Ми будемо застосовувати алгебраїчні рекурсивні типи даних для побудови системи типів, тому зручно буде також застовувати алгебраїчний підхід до генералізації теорії процесів. З алгебраїчної точки зору процесси, або кінечні автомати, представляють собою напівгрупи дій. З категоріальної точки зору процесси — це функтори які перетворюють категорії з декартовими добутками типів протоколу та стану процеса на категорію станів. Кожен процес визначається таким функтором, адже простір дій Σ для кожного процесу є унікальний.

$$
\begin{array}{lcll}
A & : & \Sigma \times X \to X & \\
X & : & \Sigma \times \Lambda_X & \\
 & | & \bot & \\
\Sigma & : & ok & \times \_ \\
 & | & error & \times \_ \\
 & | & io & \times \_ \\
 & | & \bot &
\end{array}
$$

У функціональних мовах програмування така категорія будем мати вигляд програми, де функтор буде представлений фунцією перетворення станів процесу, а протокол буде типом-сумою усіх можливих запитів до процесу:

```
  action : ( protocol, state ) -> state
   state : ( protocol, _____ )
         | []
protocol : ( ok,       _____ )
         | ( error,    _____ )
         | ( io,       _____ )
         | []
```

Визначення процесу

Щоб формально визначини процес ми спочатку визначимо домени або типи, які фігурують при визначенні процесу.

Errors	:	#error
Sucesses	:	#history
————————		————————————
Environment	:	#document | #config
Taxonomy	:	#event | #task | #flow
Module	:	$\{action : \Sigma \to P \to P\}$
————————		————————————
Process	:	#process

Напівгрупа активностей

Дана алгебраїчна модель визначає достатньо просту та визначену структуру. Мета нашої структури забезпечити детермінованість послідовності, її початковий елемент та термінальний.

$$\begin{array}{lcl} spawn & : & \top \rightarrow P \\ run & : & P \rightarrow \bot \\ \bot & : & \odot P \rightarrow P \\ complete & : & \otimes P \rightarrow P \end{array}$$

Одиниця напівгрупи демонструється номінально як операція запиту поточного стану процеса. Інша бінарна операція напівгрупи по відношеню до протокола та стану процеса *complete* передбачає виконання функції активності поточного кроку процесу, яка визначена в сигнатурі як $action : \Sigma \rightarrow P \rightarrow P$.

Алгебра процесів

Алгебра процесів визначає базові операції мультиплексування двох чи декількох протоколів в рамках одного процесу (добуток), а також паралельного та повністю ізольованого запуску включно зі стеком та областю памяті (сума) на віртуальній машині.

$$\begin{array}{lcl} \oplus & : & P \parallel P \\ \otimes & : & P \mid P \end{array}$$

Лінійність обчислень

Списки є фундаментальними структурами та найпростішим рекурсивним алгебраїчним типом. Природа обчислень, як результат виконання процессів теж лінійна. Можна навіть визначити оператор похідної, як функтор, який для алгебраїчного типу процесу буде давати список, а для списку буде давати скаляр. Як було показано [14] мережі петрі, а значить і процеси є моноїдами які генерують лінійну послідовність подій.

$$\begin{array}{lcl} \Delta\ \Lambda^0 & : & \times^0 \\ \Delta\ \Lambda^n & : & \Lambda^{n-1} \end{array}$$

Протоколи повідомлень

Простір дій протоколу визначається сумою усіх можливих повідомлень.

$$\begin{array}{lcll} App & : & init & \times\ \_ \\ & | & client & \times\ \_ \\ & | & replay & \times\ \_ \\ & | & \bot & \end{array} \qquad \begin{array}{lcll} Store & : & put & \times\ \_ \\ & | & add & \times\ \_ \\ & | & fold & \times\ \_ \\ & | & \bot & \end{array} \qquad \begin{array}{lcll} Proc & : & spawn & \times\ \_ \\ & | & complete & \times\ \_ \\ & | & run & \times\ \_ \\ & | & \bot & \end{array}$$

Лямбда числення

Для розробки нашої теорії ми застосуємо інтуітіоністську теорію типів Мартіна Льофа. Ця система типів буде застосовуватися для довення робочих теорем. База мови складає лябмда числення як внутрішня мова декартово-замкненої категорії. Лямбда числення в своїй основі скаладається з конструктора типу яка визначає функції та елімінатора – аплікації функції до аргументу.

$$\frac{\Gamma\ x : A \vdash\ B}{\Gamma \vdash fun\ x : A \to B} \qquad \frac{\Gamma\ f : A \to B \qquad \Gamma\ a : A}{\Gamma \vdash f\ a : B}$$

Алегбраїчні типи даних

Далі йде внутрішня мова алгебраїчних типів даних яка складається з суми та добутку, за допомогою яких контруюються суми-протоколи та кортежі-пакети, а також **nil** типу-атому, яким зручно термінувати рекурсивні типи даних, такі як **cons**.

$$\frac{}{\Gamma \vdash\ \bot}$$

$$\frac{\Gamma \vdash\ a : A \qquad \Gamma \vdash\ b : B}{\Gamma \vdash\ (a, b) : A \times B} \qquad \frac{\Gamma\ x : A \times B}{\Gamma \vdash fst\ x : A; \Gamma \vdash snd\ x : B}$$

$$\frac{\Gamma \vdash\ a : A \qquad \Gamma \vdash\ b : B}{\Gamma \vdash a \mid b : A \otimes B} \qquad \frac{\Gamma\ x : A \otimes B}{\Gamma \vdash inl\ x : A; \Gamma \vdash inr\ x : B}$$

Процеси і протоколи

Також ми анонсуємо процес як фундаменльний тип даних, подібний до функції але який здатний тритати певний стан у вигляді типа коротежа та бути здатним реагувати на повідомлення суми.

$$\frac{\dfrac{\Gamma\ \vdash e : Error \qquad \Gamma\ \vdash h : History}{\Gamma \vdash module : a : \Sigma \times P \to P} \qquad \Gamma\ \vdash env : Document \qquad \Gamma\ \vdash s : Event|Task|Flow}{\Gamma \vdash p : Process}$$

$$\frac{\Gamma\ \vdash \Sigma, X \qquad \Gamma\ \vdash p : Process}{\Gamma \vdash spawn_\Sigma\ p : \pi}$$

$$\frac{\Gamma\ \vdash spawn : \pi \qquad \Gamma\ \vdash m : \Sigma}{\Gamma \vdash join_\Sigma\ (m, spawn) \to \Sigma; \Gamma \vdash async_\Sigma\ (m, spawn) \to \Sigma}$$

Логіка та квантори

Далі йдуть квантори $\forall$ та $\exists$ які теж виражаються як конструкції типів:

$$\frac{\Gamma\ x : A \vdash B \qquad \Gamma\ \vdash A}{\Gamma\ \vdash \Pi(x : A)B} \qquad \frac{\Gamma\ \vdash a : A \qquad \Gamma\ x : A \vdash B \qquad \Gamma\ b : B(x = a)}{\Gamma\ \vdash (a, b) : \Pi(x : A)B}$$

$$\frac{\Gamma\ x : A \vdash B \qquad \Gamma\ \vdash A}{\Gamma\ \vdash \Sigma(x : A)B} \qquad \frac{\Gamma\ \vdash a : A \qquad \Gamma\ x : A \vdash B \qquad \Gamma\ b : B(x = a)}{\Gamma\ \vdash (a, b) : \Sigma(x : A)B}$$

$$\frac{\Gamma\ \vdash x : A \qquad \Gamma\ \vdash x' : A}{\Gamma\ \vdash Id_A(x, x')}$$

рефлексивність	:	$Id_A(a, a)$
підстановка	:	$Id_A(a, a') \to B(x = a) \to B(x = a')$
симетричність	:	$Id_A(a, b) \to Id_A(b, a)$
транзитивність	:	$Id_A(a, b) \to Id_A(b, c) \to Id_A(a, c)$
конгруентність	:	$(f : A \to B) \to Id_A(x, x') \to Id_B(f(x), f(x'))$

Контраваріантні процеси

Ко-процеси це моноїди з перевернутою стрілкою, які визначають зворотній шлях виконання події та обернену зміни стану в зворотньому порядку. З точки зоро типів нічого не змінюється, однак оберненої функції-фунтора процесу-моноїда може і не існувати. Процеси які є одночасно коваріантними та контраваріантними називаються інваріантними процесами, сигнатура яких алгебраїчно збігається з сигнатурою бінарного дерева.

$$\begin{array}{lcl} X & : & \times^i \\ \Lambda & : & \times^4\ I\ X\ \Lambda_{next}\ \Lambda_{prev} \\ & | & \bot \end{array}$$

$st(p) = p(st - 1)$ — застосування функції до протокольної заявки та стану процесу
$p - 1(st) = st - 1$ — обернена функція

Типи процесів

$action$	:	$proc_{Proc}\ (Proc \mid \Sigma) \times process \to process$
$event$	:	$proc_{App}\ (App \mid \Sigma) \times cx \to cx$
$operation$	:	$proc_{Store}\ (Store \mid \Sigma) \times container \to container$

Віртуальна машина

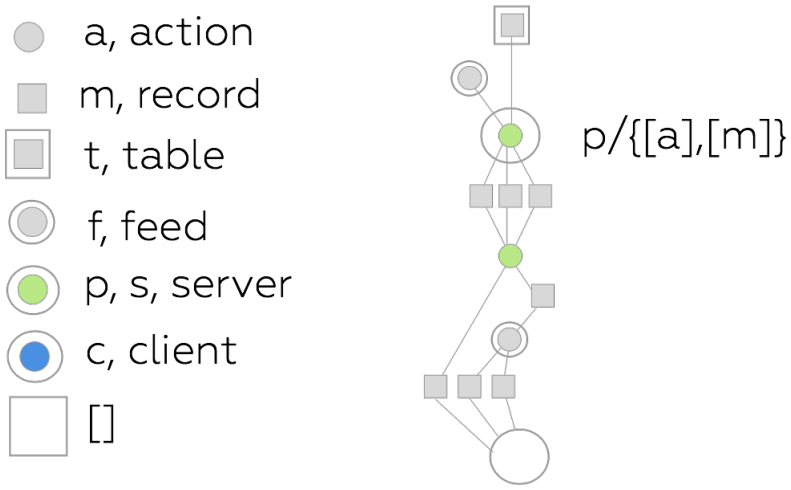


Рис. 1: Базові елементи віртуального середовища

action is the piece of code applied to message using pattern matching.
record is the tuple that defines a protocol.
client is the true customer of the system who is using service on daily basis.
server is the process nicely scheduled by the system.
feed is the server managed list of records.
table is the ixset or ets based queryable tables.

Сигнатури сервісів

app is the web clients monoidal state machine.
proc is the oriented graph monoidal executional engine.
store is the 2-category key-value database.

$$
\begin{array}{lcl}
store^3 & : & \times^3\Lambda_{id_seq}\ \Lambda_{table}\ \Lambda_{feed} \\
roster^2 & : & \times^2\Lambda_{feed}\ \Lambda^2_{feed} \\
app^2 & : & \times^2\Lambda_{session}\ \Lambda_{proc} \\
depot^1 & : & \times^1\Lambda_{account} \\
act^1 & : & \times^1\Lambda_{proc}
\end{array}
$$

1.7 Персистентність послідовностей

Для забезпечення персистентного зберігання послідовностей використовується очевидний спосіб зберігання елементів послідовності у key-value сховищі разом з референсами на наступні, та у випадку контраваріантних процесів, також і на попередні елементи у послідовності. Таким чином ми можемо перелічити усі елементи послідовності. Акумулятор розділеної у просторі та часі операції накопичує усі необхідні агрегації після кожної операції з послідовністю. Таким чином похідна від алгебраїчної структури списку буде скаляр-акумулятор після завершення перелічення.

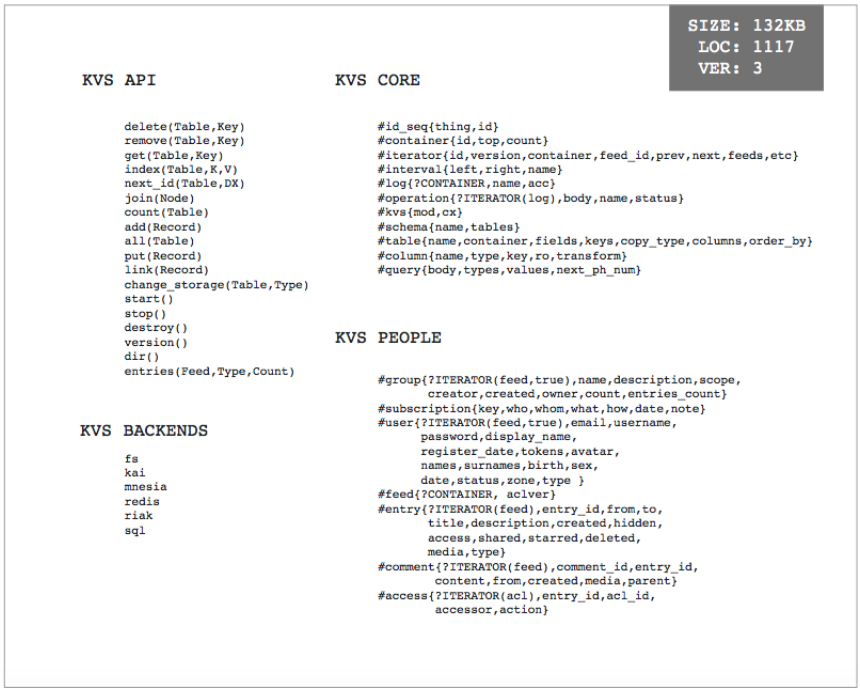


Рис. 2: Типи, функції та модулі

В даній роботі ми розглянемо формальні визначення сховища, його референсів, послідовностей через рекурсивні алгебраїчні типи даних та процеси, черги яких забезпечують впорядкованість послідовності запитів до послідовностей, що веде до лінійності операції у часі.

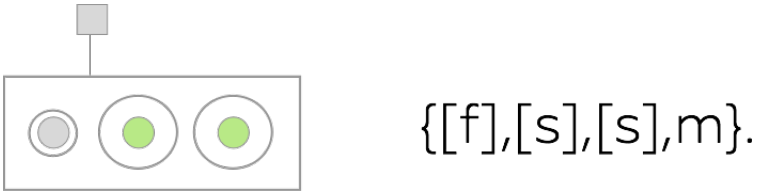


Рис. 3: Послідовності ключів, таблиць, індексів та інтерфейс

1.8 Мультипротокольный чат

Рис. 4: Типи, функції та протокол

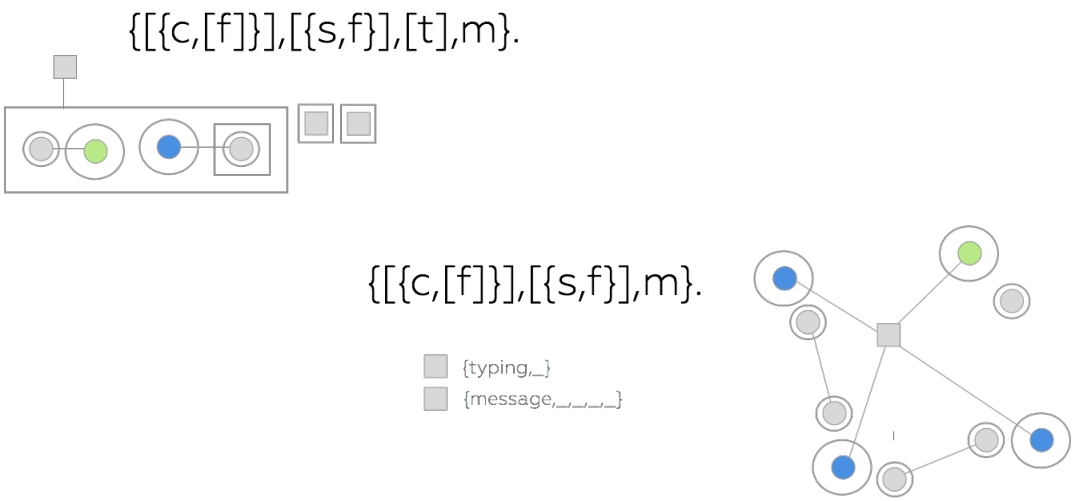


Рис. 5: Списки кімнат та користувачів і їхні чати, дві таблиці та протокол

1.9 Мета дослідження

Основною метою дослідження є збезпечення виконання критеріїв відповідності показників ефективності, детермінованості та якості шляхом впровадження результату даної теорії, комплексу програмного забезпечення у виробничий процес.
За рахунок компактності передбачається значна економія складності та простота у експлуатації, що веде до сниження собівартості впровадження. Коректність передбачає заміну автоматозаваному тестуванню, а закритість середовища дозволяє гнучко використовувати середовище на різних виробничих платформах.

1.10 Результати

Одна з останніх систем, яка була впроваджена для заміни існуючої системи, мала показники у співвідношенні розміру 10, швидкості 20 та виконання 3. Інші системи у історичній послідовності мали ці показники такими 10, 2, 8, та 19, 5, 5. В середньому показники ефективності відрізняються більше ніж на пів порядка в кращу сторону після впровадження продуктів на базі розробленої моделі. Існують продукти також і на інших мовах програмування.

Список литературы

[1] Per Martin-Löf *Intuitionistic Type Theory.* 1984

[2] J.Armstrong. *Making reliable distributed systems in the presence of sodware errors.* 2003.

[3] Robin Milner. *A Calculus of Communicating Systems.* 1986.

[4] William Lawvere. *Conceptual Mathematics.* 1997.

[5] Benjamin Pierce. *Basic category theory for computer scientist.* 2004.

[6] Сандерс Мак Лейн. *Категории для работающего математика.* 2004.

[7] Leslie Lamport. *Specifying Systems.* 2004.

[8] Michael Barr and Charles Wells. *Toposes, Triples and Theories.* 2000.

[9] Michael Barr and Charles Wells. *Category Theory for Computing Science.* 1995.

[10] И.Бакур, А.Деляну. *Введение в теорию категори и функторов.* 1972.

[11] Robin Milner. *Communicating and Mobile Systems: The π-calculus.* 1999.

[12] Robin Milner. *The Polyadic π-Calculus: A Tutorial.* 1993.

[13] Коваленко. *Теория Массового Обслужсивания.* 1965.

[14] Meseguer, Montanari. *Petri Nets Are Monoids.* 1990.

[15] Philip Wadler *Call-by-Value is Dual to Call-by-Name.* 1000

[16] D.Mostrous, V.Vasconcelos *Session Typing for a Featherweight Erlang.* 1990.

[17] S.Marlow, P.Wadler *A practical subtyping system for Erlang.* 1997

[18] A.Lindgren *A Prototype of a Soft Type System for Erlang.* 1996

[19] C. McBride. *Dependently Typed Func- tional Programs and their Proofs.* 1999

[20] C. McBride. *The Derivative of a Regular Type is its Type of One-Hole Contexts.*

[21] C. Schwarzweller. *Mizar Veri cation of Generic Algebraic Algorithms.* 1997

[22] L.Moura., S.Kong *Elaboration in Dependent Type Theory.* 1997

[23] T.Coquand, G.Huet *The Calculus of Constructions.* 1988

[24] L.Lamport *Paxos Made Simple.* 2001

[25] A.Chlipala *Certified Programming with Dependent Types* 2015

З повагою, команда Synrc.

Андрій Задорожній, генеральний директор
Сохацький Максим, технічний директор
Кирилов Володимир, співзасновник