Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Національна академія наук України
Інститут телекомунікацій і глобального інформаційного простору

Кваліфікаційна наукова праця
на правах рукопису

Захарченко Тарас Леонідович

УДК 004.021

ДИСЕРТАЦІЯ

Композитосутнісні моделі адаптивних процесональних середовищ

05.13.06 – Інформаційні технології

05 Технічні науки

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

 (Захарченко Т.Л.)

Науковий керівник Редько Ігор Володимирович, д.ф.-м.н., проф.

Київ – 2018

АНОТАЦІЯ

Захарченко Т.Л. Композитосутнісні моделі адаптивних процесональних середовищ. - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук (доктора філософії) за спеціальністю 05.13.06 – інформаційні технології. - Інститут телекомунікацій і глобального інформаційного простору, Київ, 2018.

Будь-яка людська діяльність на кожному етапі її розвитку обумовлена її інтуїтивним баченням або розумінням. Природно, що найбільш спрощене розуміння за необхідністю присутнє на перших етапах розвитку діяльності. Але поступово, з розвитком самої діяльності, еволюціонує і її розуміння. Поштовхом для наступного кроку еволюції зазвичай виступає притаманна діяльності проблемність – надмірне спрощення її розуміння. Не винятком тут є й такий важливий вид людської діяльності як інформаційно-технологічна діяльність.

Причин проблемності загалом більш ніж достатньо, але, що стосується ІТД, то тут головною серед них бачиться надто спрощене, а значить - недостатньо змістовне розуміння взаємодоповнення інформаційно-технологічних процесів (ІТП) і їх результатів. Загальна природа взаємодоповнюваності дуже складна та розкриття її в повному обсязі є, напевно, неможливим. Реальним бачиться шлях поступового прагматико-обумовленого збагачення уявлень про нього.

Домінуючу роль у цьому взаємодоповненні традиційно відіграє результат ІТД. Для цього більш ніж достатньо об'єктивних підстав. Добре відомі вражаючі результати, досягнуті насамперед у математиці, які, без сумніву, є наслідком подібного домінування. Але безпрецедентно широке поширення комп'ютерної справи в усіх областях дійсності не могло не прийти в суперечність із реаліями розвитку всіх сфер людської діяльності. Змістовно кажучи, прийняте

домінування результату ІТД як позиція чітко підтвердило, що абсолютизація будь-якого, навіть виключно важливого і продуктивного позиціонування, часто призводить до вкрай негативних результатів. Адже абсолютизація будь-чого в силу її природи замкнена в актуальних даностях предмету абсолютизації. Це свідчить про нездатність адекватно реагувати на обумовлені потенційною відкритістю об'єктивних процесів пізнання еволюційні зміни бачення дійсності. Звідси виходить, що усунення головної причини недоліків у рамках актуально замкненої точки зору на ІТД є неможливим. Актуально замкнене загальнозначуще ядро розглядів як їх логіка повинно бути адекватно збагачене потенційно відкритим різноманіттям його предметних продовжень. Тобто традиційні сьогодні, актуально замкнені у своїй відкритості або замкненості парадигми ІТД мають бути якісно збагачені відповідно до їх взаємодоповнення – відкрито-замкненої парадигми. Змістовно така зміна бачиться у переході в ІТД від продукування окремих, іноді досить ефективних, але фактично обмежених домінантою результату ІТД підходів до вирішення задач, до розробки єдиної відкрито-замкненої інформаційно-технологічної платформи, домінантою якої є не результат, а ІТП. Це дозволяє не протиставляти існуючі точки зору на ІТД, а навпаки, природним чином об'єднувати їх у якості тих чи інших предметних продовжень «спільного знаменника» – взаємодоповнення ІТП та їх результатів. Така об'єднуюча точка зору складає кістяк реального розуміння ІТД. Вона дозволяє, відштовхуючись від фундаментального значення інтуїтивної бази сучасної ІТД, якісно його розвивати, доповнюючи за можливістю точними дослідженнями та розробками.

У дисертації в якості такого «спільного знаменника» різних точок зору на ІТД пропонується концепція адаптивного процесонального середовища (АПС). Основу його складає адекватне зміщення акцентів розгляду з виключно результатів вирішення задач, зокрема їх верифікації на коректні методи їх досягнення. Це забезпечує можливість ставити та ефективно вирішувати задачі управління *якістю* отримуваних рішень, забезпечення *ефективності* діяльності

з пошуку таких рішень і проблему збереження *інвестицій* у такій діяльності. Тут під якістю розуміється коректність продукованих АПС рішень, їх гарантоздатність та функціональна безпека. Під ефективністю ж розуміється підвищення продуктивності розробки за рахунок простоти використання існуючих суміжних інформаційно-технологічних рішень і уточнення самої ІТД за рахунок підтримки АПС, прийняття інформаційно-технологічних рішень в умовах часткової невизначеності. Отримати рішення цих та інших задач стало можливим завдяки залученню до розглядів, у першу чергу, процесів їх вирішення, зокрема адекватній організації цих процесів. Останнє забезпечується реальним врахуванням активної участі суб'єкта в ІТП, там, де ця участь дійсно необхідна.

Основу ІТД складає його розуміння як обумовлення результату ІТД при цьому результат ІТД є сутністю (монадою), а ІТД є процесом розкриття її суті (концепту). Таке відношення між суттю та сутністю називається сутесутнісним відношенням. У цього відношення можна виділити властивості рефлексивності, транзитивності та антисиметричності. В силу приведених властивостей ІТД можна звести до рефлексивно-транзитивного замикання сутесутнісного відношення. Звісно, для того, щоб говорити про ІТД цього недостатньо, необхідно збагатити це відношення до необхідного рівня. В роботі ці збагачення проходять починаючи з розгляду динамічного та статичного аспектів ІТД, закінчуючи втягненням до розглядів таких понять як результат ІТД, композиція та функція. А саме ССВ зводиться до композитосутнісного відношення (КСВ). Це означає що суттю сутності в нашому специфічному випадку (ІТД) виступає композиція. В роботі ґрунтовно досліджено поняття композиції, зокрема, виділено основні їх властивості: адекватність, тотальність, замкненість. Особливо важливою є адекватність, як носій суб'єктивності в ІТД. Для більш глибокого розуміння композицій розроблені їх класифікації: за роллю та властивостями даних. Відомо три ролі композицій: аплікативні, фундативні та

константні. За властивостями даних вони поділяються на абстрактні, іменні та метаіменні.

Виходячи з того, що КСВ – частковий порядок, ми можемо сформулювати та довести прагматико-обумовлену спеціалізацію теореми Тарського: проективні функції мають нерухомі точки. Тут проективною функцією називається та функція, що забезпечує ітеративний перехід від задачі до рішення. Принципова значимість цієї теореми полягає, звичайно, не в доведенні існування нерухомих точок у проективних функцій, а в тому, що вона відкриває зовсім природній шлях для пошуку тих типів прагматико-обумовлених функцій, для яких їхнє існування є більш-менш прямим наслідком такої обумовленості.

Зважаючи на те, що у ІТД можна виділити певну загальнозначущу логіку – рефлексивно-транзитивне замикання КСВ, та відкрите різноманіття прагматично-обумовлених її продовжень, з’явився сенс говорити про відкирто-замкнену парадигму ІТД, яка лягла в основу АПС. Базовим поняттям відкрито-замкненої парадигми стало відкрито-замкнене середовище (ВЗСр), яке включає в себе згадані вище відкриту та замкнуту частини. Дані побудови все ще досить загальні, а розуміння АПС як ВЗСр не достатньо. В роботі проводиться ряд збагачень ВЗСр, які приводять до розуміння АПС, як прагматико-обумовленої релятивізації універсального середовища інтеграції. Таке середовище має наступну архітектуру: УСІ = <<< УР, УП>, УЗП>, ЗЕР>. Вона побудована на взаємодоповненні універсуму рішень (УР) та універсуму предметів(УП, композиції у пасивній ролі), що доповнюють універсумі засобів побудови (УЗП, композиції в активній ролі), це доповнення в свою чергу доповнює засоби еволюційного розвитку (ЗЕР, метакомпозиції). Характер цих взаємодоповнень, як можна помітити, носить суб’єктно-об’єктний характер та дозволяє говорити про конкретні реалізації АПС.

Подальші дослідження АПС виявляються неможливими без актуалізації алгебри яка використовується в ІТД. Вона актуалізована як примітивна програмна алгебра (ППА). Її зручно використовувати в розробці програмного

забезпечення, так як її композиції нагадують управляючі структури класичних мов програмування. Також вона може використовуватись і для розробки апаратного забезпечення завдяки властивості кортежності носіїв алгебри. Це дозволило дослідити такий актуальний носій, як функції над записами. Розглянуті підкласи функцій що зберігають та не зберігають денотати і для них вирішені проблеми повноти. В результаті розглядів отримав подальший розвиток метод отримання алгебраїчних характеристик класів функцій.

Взаємодія розробника з АПС неможлива без мови специфікації композицій(МСК). У роботі запропонована як текстова, так і графічна МСК. Вони підтримують, як породження композицій (метакомпозиції), так і застосування їх.

З метою досліджень, в ході роботи розроблена дослідна реалізація адаптивного процесонального середовища. Воно підтримує розробку як апаратного, так і програмного забезпечень. Користувач може задати дескрипцію вирішення задачі у МСК. Після цього вирішення може бути згенероване в код HDL або на мові програмування. На разі підтримуються Verilog HDL та C. У випадку якщо мова йде про розробку апаратного забезпечення, то рішення виглядає як Verilog модуль, що сумісний з САПР Quartus II.

Ключові слова: Адаптивне процесональне середовище, композиція, відкрито-замкнене середовище, сутесутнісне відношення, композитосутнісне відношення.

Список публікацій здобувача

[1] T. L. Zakharchenko, "Adaptive hardware design," *Технологический аудит и резервы производства,* no. 5, pp. 23-31, 2015.

[2] T. Zakharchenko, "Pragmatics dependent hardware design," in *Proceedings of the 2014 IEEE 34th International Scientific Conference "Electronics and nanotechnology"*, Kyiv, 2014.

[3] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» *Вісник КНУТД,* № 1, pp. 35-39, 2014.

[4] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» в *Збірник статей VI Міжнародної науково-технічної конференції молодих вчених «Електроніка-2013»*, Київ, 2013.

[5] Т. Л. Захарченко, «Вирішення проблеми повноти в композиційному програмуванні,» в *Матеріали XIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики»*, Київ, 2015.

[6] Т. Л. Захарченко, «Швидкодіючий багатошаровий персептрон, що гнучко масштабується». Україна Патент 72313, 10 08 2012.

[7] Т. Л. Захарченко, І. В. Редько, Д. І. Редько та П. О. Яганов, «Примітивна програмна алгебра обчислюваних функцій над записами,» *Наукові вісті НТУУ «КПІ»,* № 2, pp. 29-40, 2015.

[8] D. I. Redko, I. V. Redko, P. O. Yahanov and T. L. Zakharchenko, "Compositional basis in programmer activity," *Системні дослідження та інформаційні технології,* no. 4, pp. 83-96, 2015.

[9] Т. Л. Захарченко, «Проблема повноти в класі функцій над записами, які зберігають денотати,» *Наукові вісті НТУУ «КПІ»,* № 5, pp. 23-30, 2015.

[10] І. В. Редько, Д. І. Редько та Т. Л. Захарченко, Концептологічні основи проектування, Київ: Компринт, 2016.

Taras Zakharchenko. Composition-gist models of adaptive processional environments – Manuscript.

Dissertation research for degree of PhDs on speciality 05.13.06 information technologies.- Institute of Telecommunications and Global Information Space of the National Academy of Sciences of Ukraine, Kyiv, 2018.

Any human activity on every stage of its development conditioned by its intuitive vision and understanding. Naturally that the most simplified understanding present on earlier stages of the activity's development. But with development of the activity, its understanding also evolves. In the most cases the push to the next step of evolution is the problematic of the activity – oversimplification of the activity's understanding. The information-technologic activity (ITA), as important kind of human activity, is not exception here also.

It is more than enough reasons of the problematic, but regarding to the ITA, the main reason is oversimplified, hence not enough informative, understanding mutual supplementation of IT-processes and their results. The nature of the mutual supplementation is very complicated and it is impossible to fully reveal it. More real way is gradual pragmatically-conditioned enrichment of its conception.

The domination role in this mutual supplementation is traditionally played by the result of ITA. It is more than enough reasons for this. Well known impressive results achieved mostly in mathematics, are consequence of the domination. But unprecedentedly wide spread of information technologies in all domains of reality, came to contradiction with realities of all human activity domains development. In other words, adopted as position, result's domination confirmed that absolutization of anything, even if the thing is exceptionally important, leads to very negative results. This is because that absolutization closed in its actualities of the subject, due to its nature. This is evidence of thing that one unable to adequately react on evolution changes of reality vision, conditioned by potential openness of cognitive processes. Hence, it is impossible to eliminate the main reason of the shortcomings in the framework of actually closed point of view on the ITA. Actually closed kernel of views as their logic must be adequately enriched by the potentially open diversity of its

subjective sequels. In the other words, traditional ITA paradigms, which are actually closed in their openness/closeness must be enriched according to their mutual supplementation – open-closed paradigm (OC-paradigm). We see such change in the transition from production of separate, sometimes quite effective, but limited by domination of ITA's result, approaches to ITA of the united open-closed ITA platform, where dominates ITA as activity targeted on result(solution) creation. This allows to not put existing points of view on ITA in opposition to each other, but naturally to unite them as subject sequels of the "common denominator" – mutual supplementation of the IT-processes and their results. Such uniting point of view is framework of the real understanding of the ITA. It allows starting from fundamental significance of intuitive basis of the modern ITA, quality develop it by supplementing it, when it is possible, with precise researches and developments.

In the paper as such "common denominator" of different points of view on ITA a concept of adaptive processional environment (APE) proposed. Basis of the APE consists of adequate shift of accents from considering solely results of problem's solutions, including verification to actual methods of the solutions obtaining. This provides possibility to put and effectively solve problems of quality management and effectiveness assurance of the solution finding. The problem of investments preservation is also solved. Here we understand quality as correctness of produced by APE solutions, their reliability and functional safety. The effectiveness is the ITA performance increase because of simplicity of use of already existing adjacent solutions and specification of the IT-process itself with support APE and taking decisions in conditions of partial uncertainty. It is become possible to obtain solutions of this or another problems by involving processes of their solution and adequate organization of the processes in the considerations. Adequate organization of the IT-processes is achieved by taking in account active role of the designer in the process, where designer's decisions are really needed.

The basis of the ITA is its understanding as conditioning of the designed thing, where designed thing is substance and the ITA is a process of revealing its gist

(concept). Such relations between the substance and the gist are named “substance-gist relations” (SGR). There are three properties of the relation found: reflectivity, transitivity and antisymmetricity. Because of listed properties the ITA can be reduced to reflective-transitive closure of SGR. Of course, this is not enough to talk about ITA, this relation should be enriched to necessary level. In the paper the enrichment conducted starting from consideration of dynamic and static aspects of the ITA to consideration such concepts as solution, composition and function. Namely, SGR is reduced to composition-gist relation (CGR). This means that gist of the substance in our specific case (ITA) is composition. In the paper concepts of composition profoundly studied, also marked out composition’s properties: adequacy, totality, closedness. Particularly important is adequacy, it is carrier of subjectivity in the ITA. For more profound understanding of compositions, their classifications are specified: by role and by data properties. There are three types of compositions by roles: applicative, fundative and constant. And there are three types of compositions by data properties: abstract, named, meta-named.

Because of the fact that CGR is partial order, we can formulate and prove pragmatically conditioned theorem of Tarsky: projective functions have fixed points. Here term “projective function” means function that provides iterative transition from problem to solution. Principal importance of the theorem is not in the proof of existence of fixed points, but in the thing that it opens fully natural way for pragmatically conditioned functions search, for which their (functions’) existence is more or less direct consequence of such conditioning.

Due to fact that in the ITA we can mark out some common logic – reflective-transitive closure of CGR and open diversity of pragmatically-conditioned sequels of the logic, it makes sense to discuss OC ITA paradigm which is foundation of APE. The essential concept of OC-paradigm is OC-environment (OCE), which include mentioned above open and closed parts. Given results are still too general and it is not enough to understand APE as OC-environment. Series of OCE enrichments conducted in the paper, so they led to understanding APE as pragmatically conditioned

relativisation of universal integration environment (UIE). Such environment has following architecture: UIE=<<<SolU, SU>, DMU,EMU>. It is based on: mutual supplementation of solution universe (SolU) and subject universe (SU, compositions in passive role) which altogether supplement ITA means universe (DMU, compositions in active role), this supplementation supplements means of evolution universe (EMU, metacompositions). The nature of these mutual supplements allows us to talk about certain implementation of APE.

Further research of APE is impossible without actualization of IT-algebra. It is actualized as primitive program algebra (PPA). It is handy to use it in SW design because its compositions resemble statements in programming languages. Also, it may be used for HW design due to tupleness of the algebra's carriers. This allowed studying such actual carrier as functions on records. There are two subclasses of the functions considered: functions which preserve denotates and which are not. One of the results in the research is method of algebraic characteristics obtaining of different function classes.

Interaction of the designer with APE is impossible without composition specification language (CSL). In the paper text and graphical CSLs are proposed. They are supporting both producing(metacompositions) and application of compositions.

Due to research purposes, during the work on the paper, implementation of APE was proposed. It supports design both HW and SW. The user may define description of problem solution with CS. After this, the solution may be translated to HDL code or some software source code. Verilog HDL and C are supported for now. In the case of HW design, the solution looks like Verilog module which conforms Quarus II design software.

Keywords: Adaptive processional environment, composition, open-closed environment, subject-gist relation, composition-gist relation.

Publications list of the candidate

[1] T. L. Zakharchenko, "Adaptive hardware design," *Технологический аудит и резервы производства,* no. 5, pp. 23-31, 2015.

[2] T. Zakharchenko, "Pragmatics dependent hardware design," in *Proceedings of the 2014 IEEE 34th International Scientific Conference "Electronics and nanotechnology"*, Kyiv, 2014.

[3] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» *Вісник КНУТД,* № 1, pp. 35-39, 2014.

[4] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» в *Збірник статей VI Міжнародної науково-технічної конференції молодих вчених «Електроніка-2013»*, Київ, 2013.

[5] Т. Л. Захарченко, «Вирішення проблеми повноти в композиційному програмуванні,» в *Матеріали XIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики»*, Київ, 2015.

[6] Т. Л. Захарченко, «Швидкодіючий багатошаровий персептрон, що гнучко масштабується». Україна Патент 72313, 10 08 2012.

[7] Т. Л. Захарченко, І. В. Редько, Д. І. Редько та П. О. Яганов, «Примітивна програмна алгебра обчислюваних функцій над записами,» *Наукові вісті НТУУ «КПІ»,* № 2, pp. 29-40, 2015.

[8] D. I. Redko, I. V. Redko, P. O. Yahanov and T. L. Zakharchenko, "Compositional basis in programmer activity," *Системні дослідження та інформаційні технології,* no. 4, pp. 83-96, 2015.

[9] Т. Л. Захарченко, «Проблема повноти в класі функцій над записами, які зберігають денотати,» *Наукові вісті НТУУ «КПІ»,* № 5, pp. 23-30, 2015.

[10] І. В. Редько, Д. І. Редько та Т. Л. Захарченко, Концептологічні основи проектування, Київ: Компринт, 2016.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 15
ВСТУП 16
РОЗДІЛ 1. ЗАГАЛЬНОЗМІСТОВНІ ПЕРЕДУМОВИ АДАПТИВНИХ ПРОЦЕСОНАЛЬНИХ СЕРЕДОВИЩ 23
1.1 Основні принципи ІТ-діяльності 24
1.1.1 Функціональна платформа 27
1.1.2 Об'єктно-орієнтована платформа 28
1.1.3 Логічна платформа 29
1.1.4 Структурна платформа 30
1.1.5 Модульна платформа 31
1.1.6 Денотаційна платформа 31
РОЗДІЛ 2. ТЕОРЕТИЧНІ ЗАСАДИ АПС 34
2.1 Збагачення розуміння ІТД 34
2.1.1 Сутесутнісна природа розуміння ІТД 34
2.1.2 Композитосутнісне збагачення розуміння ІТД 37
2.1.3 Відкрито-замкнені аспекти ІТД 45
2.2 Композитосутнісні засади АПС 51
2.2.1 Денотативні аспекти АПС 51
2.2.2 Композитосутнісні аспекти АПС 53
2.2.3 Композитосутнісне відношення – ядро АПС 56
2.2.4 Інтуїтивні властивості ІТ-рішень 63
2.3 Адаптивні середовища для вирішення ІТ-задач 65
2.3.1 Підходи до розробки реконфігурованих систем 70
2.4 Композиційні засади адаптивних процесональних середовищ 75
2.4.1 ІТ-задача, функція та композиція в першому наближенні 76
2.4.2 Аплікативні композиції 93
2.4.3 Фундативні композиції 102
РОЗДІЛ 3. СЕМАНТИЧНІ ДОСЛІДЖЕННЯ АПС 116
3.1 ППА в класі функцій над записами, що не зберігають денотати 116
3.1.1 Загальні положення 116
3.1.2 Повнота в класах чр-функцій та чр-предикатів. Критерій повноти 120
3.1.3 ПА чр-функцій та чр-предикатів над множиною записів 124
3.2 ППА в класі функцій над записами, що зберігають денотати 132
3.2.1 Загальні положення, визначення та позначення 132
3.2.2 Повнота ППА у класі чр-функцій, що зберігають денотати та чр-предикатів над записами 137
3.3 Семантичний аналіз мови Verilog 143
3.3.1 Загальні положення мови Verilog 143
3.3.2 Композиції у мові Verilog 144
РОЗДІЛ 4. ІМПЛЕМЕНТАЦІЯ АДАПТИВНОГО ПРОЦЕСОНАЛЬНОГО СЕРЕДОВИЩА 152

4.1 Мови специфікації композицій 152
4.2 Архітектурні особливості реалізації АПС 156
4.3 Інтеграція композиційного апаратного забезпечення з обчислювальними системами 158
4.3.1 Verilog модулі адаптивного апаратного забезпечення 158
4.3.2 Драйвери для адаптивного апаратного забезпечення 164
4.4 Аналіз ефективності АПС 166
ВИСНОВКИ 173
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 175
ДОДАТОК А. АКТИ ВПРОВАДЖЕННЯ 186
ДОДАТОК Б. СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ 189
ДОДАТОК В. ВІДОМОСТІ ПРО АПРОБАЦІЮ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ 191

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АПС – адаптивне процесональне середовище

ВЗС – відкрито-замкнута система

ВЗСр – відкрито-замкнуте середовище

ЗЕР – засіб еволюційного розвитку

ІТС – інформаційно-технологічні системи

ІТСП – інформаційно-технологічно-сутнісна платформа

ІТД – інформаційно-технологічна діяльність

КЛ – композиційна логіка

КСВ – композитосутнісне відношення

ЛП – логічна платформа

ЛПВ – логіко-предметне відношення

ППА – примітивні програмні алгебри

САТ – середовище адаптивних типізацій

СП – структурна платформа

ССВ – сутесутнісне відношення

ТОП – типізація обумовлена прагматикою

УЗП – універсум засобів побудови

УП – універсуму предметів

УР – універсум рішень

УСІ – універсальне середовище інтеграції

ФП – функціональна платформа

DSP – Digital Signal Processor

FPGA – Field Programming Gate Array

HDL – Hardware Description Language

PBD – Platform Based Design

TLM – Transaction Level Modelling

ВСТУП

Будь-яка людська діяльність на кожному етапі її розвитку обумовлена її інтуїтивним баченням або розумінням. Природно, що найбільш спрощене розуміння за необхідністю присутнє на перших етапах розвитку діяльності. Але поступово, з розвитком самої діяльності, еволюціонує і її розуміння. Поштовхом для наступного кроку еволюції зазвичай виступає притаманна діяльності проблемність – надмірне спрощення її розуміння. Не винятком тут є й інформаційно-технологічна діяльність (ІТ-діяльність, ІТД), пов’язана з індукованими нею інформаційно-технологічними процесами (ІТП).

Причин проблемності загалом більш ніж достатньо, але, що стосується ІТД, то тут головною серед них бачиться надто спрощене, а значить – недостатньо змістовне розуміння взаємодоповнення відповідних процесів та їх результатів. Загальна природа взаємодоповнюваності дуже складна та розкриття її в повному обсязі є, напевно, неможливим. Реальним бачиться шлях поступового прагматико-обумовленого збагачення уявлень про нього.

Домінуючу роль у цьому взаємодоповненні традиційно відіграють результати ІТД. Для цього більш ніж достатньо об'єктивних підстав. Добре відомі вражаючі результати, досягнуті, насамперед, у математиці, які, без сумніву, є наслідком подібного домінування. Але безпрецедентно широке поширення комп'ютерної справи в усіх областях дійсності не могло не прийти в суперечність із реаліями розвитку всіх сфер людської діяльності. Змістовно кажучи, прийняте домінування результатів ІТД як позиція чітко підтвердило, що абсолютизація будь-якого, навіть виключно важливого і продуктивного позиціонування, часто призводить до вкрай негативних результатів. Адже абсолютизація будь-чого в силу її природи замкнена в актуальних даностях предмету абсолютизації. Це свідчить про нездатність адекватно реагувати на обумовлені потенційною відкритістю об’єктивних процесів пізнання еволюційні зміни бачення дійсності. Звідси виходить, що усунення головної причини недоліків у рамках актуально замкненої точки зору на ІТД є неможливим. Актуально замкнене загальнозначуще ядро розглядів як їх логіка повинно бути

адекватно збагачене потенційно відкритим різноманіттям його предметних продовжень. Тобто традиційні сьогодні, актуально замкнені у своїй відкритості або замкненості ІТ-парадигми мають бути якісно збагачені, відповідно, до їх взаємодоповнення – відкрито-замкненої парадигми. Змістовно така зміна бачиться у переході в ІТД від продукування окремих, іноді досить ефективних, але фактично обмежених домінантою результату ІТД підходів до вирішення задач, до розробки єдиної відкрито-замкненої ІТ-платформи, домінантою якої є не результат ІТД, а ІТД як діяльність, що націлена на процеси вирішення ІТ-задач. Це дозволяє не протиставляти існуючі точки зору на ІТ-діяльність, а навпаки, природним чином об'єднувати їх у якості тих чи інших предметних продовжень «спільного знаменника» – взаємодоповнення ІТП та їх результатів, породжених в рамках ІТД. Така об'єднуюча точка зору складає кістяк реального розуміння ІТД. Вона дозволяє по-новому поглянути на такий вид ІТД, як побудова апаратного та програмного забезпечень та доповнити його точними дослідженнями та розробками.

У дисертації в якості такого «спільного знаменника» різних точок зору на ІТД пропонується концепція адаптивного процесонального середовища (АПС). Основу його складає забезпечення гнучкості ІТД шляхом адекватного зміщення акцентів розгляду з виключно результатів вирішення задач, зокрема їх верифікації на коректні методи їх досягнення. Це забезпечує можливість ставити та ефективно вирішувати задачі управління *якістю* отримуваних рішень, забезпечення *ефективності* діяльності з пошуку таких рішень і проблему збереження *інвестицій* у такій діяльності. Тут під якістю розуміється коректність продукованих АПС рішень, їх гарантоздатність та функціональна безпека. Під ефективністю ж розуміється підвищення продуктивності вирішення ІТ-задач за рахунок простоти використання існуючих суміжних рішень і уточнення самого ІТП за допомогою концепції АПС, прийняття рішень в умовах часткової невизначеності. Отримати рішення цих та інших задач стало можливим завдяки залученню до розглядів, у першу чергу, процесів їх вирішення, зокрема адекватній організації цих процесів. Останнє забезпечується реальним

врахуванням активної участі суб'єкта в ІТП, там, де ця участь дійсно необхідна. Із зазначеного безпосередньо виходить, що створення згаданого адаптивного процесонального середовища є **актуальною та важливою** задачею.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота відповідає напрямкам досліджень кафедри конструювання електронно-обчислювальної апаратури НТУУ «КПІ», а її результати отримані в рамках науково-дослідної роботи «Прискорення обчислень з використанням пристроїв, що реконфігуруються» (РК №0113U001874) згідно з основними науковими напрямками діяльності Національного технічного університету України «Київський політехнічний інститут» та пріоритетного напрямку розвитку науки і техніки України «Інформаційні та комунікаційні технології» .

Мета і завдання дослідження. Метою дисертаційної роботи є підвищення якості та ефективності ІТД шляхом розробки методу вирішення задач, що продукує композитосутнісні моделі та взаємодоповнює існуючі точки зору на ІТД й враховуючи як традиційні, так і прагматико-обумовлені, нетрадиційні аспекти ІТД. З урахуванням основних вимог до композитосутнісних моделей АПС і у відповідності до зазначеної мети в дисертації ставляться і вирішуються наступні задачі:

- розробка та обґрунтування концепції адаптивного вирішення задач та створення на її базі відкрито-замкнених засад АПС;
- побудова на базі відкрито-замкнених засад АПС логіко-предметної понятійної системи АПС та створення на цій основі композиційних засад АПС;
- створення на основі композиційних засад АПС композитосутнісної понятійної системи, основу якої складає поняття композитосутнісного відношення;
- прагматико-обумовлена типізація композицій і розробка методу композитосутнісної релятивізації;

- дослідження семантичних характеристик репрезентативних класів обчислювальних функцій над зліченними носіями;
- розробка та аналіз дослідної реалізації АПС, створеного на базі композитосутнісної моделі АПС.

Об'єкт дослідження дисертаційної роботи – інформаційно-технологічні процеси вирішення задач.

Предмет дослідження дисертаційної роботи – композитосутнісні моделі АПС у застосуванні до процесів розробки апаратного та програмного забезпечень.

Методи дослідження – проведені в дисертаційній роботі дослідження базуються на сукупності загальнометодологічних, логіко-епістемологічних, логіко-математичних методів. Серед методологічних основними є *метод сутесутнісної релятивізації* і його композитосутнісна спеціалізація, серед логіко-епістемологічних – уведення та виключення абстракції. До використаних логіко-математичних методів відноситься композиційний метод вирішення задач й алгебраїчні методи дослідження. Також використано семантичний аналіз.

Наукова новизна одержаних результатів, що виноситься на захист, полягає в наступному:

- вперше створено композиційні та композитосутнісні засади, композитосутнісну понятійну систему АПС і доведено теорему про нерухому точку проективної функції, що дозволило сформулювати як теоретичний фундамент АПС, так і композитосутнісний метод розробки апаратного та/або програмного забезпечень, що складає основу адаптивного вирішення задач на практиці.
- вперше встановлено прагматико-обумовлену типізацію композицій, що змістовно збагачує ключове в роботі поняття композиції;
- отримав подальший розвиток метод отримання алгебраїчних характеристик класів обчислюваних функцій, що забезпечило

можливість дослідження АПС, зокрема вирішення проблеми повноти в репрезентативних класах обчислювальних функцій;

- вперше виявлені семантико-синтаксичні структури HDL-мов, що дозволило визначити їх структурно-функціональну модель і скласти єдиний фундамент для розробки в рамках АПС апаратного забезпечення.

Таким чином, у дисертаційній роботі вперше висунута та розроблена концепція композитосутнісних моделей АПС, яка дозволяє розробляти засоби вирішення ІТ-задач, що адекватно відображають природу цих задач і є теоретичною основою для практичної імплементації АПС.

Практичне значення одержаних результатів визначається зокрема:

- розробленою дослідною реалізацією АПС, яка підтримує наскрізне інтерактивне вирішення задач, починаючи з властивої їм прагматики та закінчуючи автоматичною генерацією коду рішення в формальних мовах Verilog HDL та C, що дозволило дослідити практичне застосування АПС;
- виділеними практично важливими класами функцій над записами та вирішеними проблемами їх повноти в примітивних програмних алгебрах;
- створеними графічною та текстовою мовами специфікації композицій і вирішеним із їх використанням набором репрезентативних задач, що дало змогу продемонструвати ефективність вирішення ІТ-задач в АПС.

Основним і найзагальнішим практичним результатом є розроблена композитосутнісна модель адаптивного процесонального середовища, що дозволяє коректно ставити та вирішувати проблеми управління якістю рішень і забезпечення ефективності розробки програм.

Одержані в дисертації нові результати знайшли застосування в організації ТОВ «Відео Інтернет Технології» (м. Київ), використані при виконанні НДР НТУУ «КПІ», а також у навчальному процесі кафедри конструювання електронно-обчислювальної апаратури (КЕОА) факультету електроніки НТУУ «КПІ», що підтверджується відповідними актами:

− акт від 04.02.2016 про використання в ТОВ «Відео Інтернет Технології» дослідної реалізації АПС, орієнтованої на розробку програмного забезпечення для обробки зображень;

− акт від 02.12.2016 про використання в результатах НДР (РК №0113U001874) дослідної реалізації АПС для побудови на її базі надійних і ефективних засобів розробки апаратного забезпечення;

− акт від 02.12.2016 про впровадження в навчальний процес кафедри КЕОА факультету електроніки НТУУ «КПІ» дослідної реалізації АПС в курсах "Основи побудови інформаційно-обчислювальних засобів інтеграції", "Експертні системи" на практичних роботах студентів, пов'язаних із програмними алгебрами.

Особистий внесок здобувача. Основні ідеї та наукові результати дисертаційної роботи отримані автором особисто та висвітлені у десяти працях**,** шість з яких виконані одноосібно [1] [2] [3] [4] [5] [6].

У роботах, опублікованих у співавторстві, здобувачу належать:

- [7] – формування базового набору частково-рекурсивних функцій і предикатів над множиною записів, теорема про те, що цей набір складає породжуючу систему ППА;
- [8] – визначення I_m^n-базису чр-функцій і чр-предикатів над записами, побудова відображень елементів множини мультимножин на елементи множини записів, теорема про те, що цей базис складає допустиму систему ППА;
- [9] – визначення I_m^n-базису чр-функцій та чр-предикатів над записами, що зберігають денотати, побудова відображень елементів множини векторів на елементи множини записів, теорема про те, що цей набір складає породжуючу систему ППА;
- [10] –Розділи 1-5, 7, 6.2.

Апробація роботи. Наукові та практичні результати дисертаційної роботи доповідались і обговорювались на міжнародних та вітчизняних науково-технічних семінарах і конференціях:

- Всеукраїнській науково–практичній конференції «Innovations in science and technology 2012»;
- IEEE 34th International Conference on Electronics and Nanotechnology (ELNANO 2014);
- Міжнародній конференції молодих вчених «Електроніка-2012»;
- Міжнародній конференції молодих вчених «Електроніка-2013»;
- XIII Всеукраїнській науково-практичній конференції студентів, аспірантів і молодих вчених «Теоретичні та прикладні проблеми фізики, інформатики та математики».

Публікації. За темою дисертації опубліковано **10** наукових праць: 5 в українських фахових виданнях, кожна з яких індексується щонайменше в двох міжнародних наукометричних базах даних – РИНЦ і EBSCO [1] [3] [6] [7] [8] [9]; одна стаття в працях міжнародних наукових конференцій, включених у наукометричну базу даних IEEEXplore Digital Library [2]; а також дві тези доповідей на міжнародній і всеукраїнській науково-технічних конференціях [4] та [5]; патент України на корисну модель № 72313 «Швидкодіючий багатошаровий персептрон, що гнучко масштабується» [6]; монографія “Концептологічні основи проектування” [10].

Структура та обсяг роботи. Дисертаційна робота складається з переліку умовних позначень, вступу, чотирьох розділів, висновків, списку використаних джерел загальний обсяг дисертації становить 164 стор.

РОЗДІЛ 1. ЗАГАЛЬНОЗМІСТОВНІ ПЕРЕДУМОВИ АДАПТИВНИХ ПРОЦЕСОНАЛЬНИХ СЕРЕДОВИЩ

Сьогодні стан справ в інформаційно-технологічної галузі і, зокрема в створенні систем, що підтримують ІТД, зокрема розробку апаратного та програмного забезпечень, в значній мірі визначається процесом дедалі глибшого і, разом з тим, широкого проникнення її в усі сфери життя. Природно, що з кожним кроком у цьому напрямку, за необхідністю, зростають вимоги, що висуваються, як до якості продукту, що виробляється, так і до ефективності його виробництва. І, незважаючи на те, що інформаційно-технологічна діяльність сьогодні рясніє вражаючими результатами, які «говорять самі за себе», стає все більш очевидним, що ці результати носять у переважній мірі явно виражений екстенсіальний характер і підтримка зазначених вимог у цих реаліях стає все більш проблематичною, а у доступній для огляду перспективі - неможливою. Звісно, таке становище обумовлено тим, що інформаційно-технологічна діяльність, зокрема розробка АтаПЗ, й сьогодні розвивається, головним чином, на номінальній поверхнево-інтуїтивної основі. Поверховість розуміється в тому сенсі, що інтуїція як джерело реальних розглядів підміняється окремими спрощеними уявленнями про неї суб'єктів такої діяльності. Останні ж, в силу своєї суб'єктивної природи, є вже джерелом не реальних, а номінальних розглядів. Яскравим прикладом тут виступає галузь розробка програмного та апаратного забезпечення.

Об'єктивна відкритість інтуїції, а значить - і розуміння діяльності з розробки апаратного та програмного забезпечень, як виду ІТД, зумовлює принципову безперспективність точки зору на неї як на «просту суму» номінальних уявлень. Це підтверджується відомим епістемологічним результатом, отриманим ще Аристотелем, про незвідність цілого до суми його складових частин. Для забезпечення реальної основи розглядів ІТД об'єктивно необхідна зміна парадигм у цьому розвитку. Прояснити окремі важливі аспекти сформованого в ІТД положення, проаналізувати основні причини зазначеного

реально-номінального протиріччя і накреслити шляхи його розв'язання – основна мета цього розділу роботи.

1.1 Основні принципи ІТ-діяльності

Побудова середовища, яке б реально підтримувало ІТП безпосередньо пов'язана з формуванням концептуальноєдиної точки зору на саму ІТД. Причому такої, яка б не була замкненою в актуальності будь-якого окремого, навіть досить продуктивного, розуміння ІТД. Адже саме незамкненість розуміння ІТД в актуальності забезпечує можливість її еволюційного розвитку разом із ІТД, підтримує максимально широке розмаїття розумних інтуїтивних уявлень про нього. Така побудова потребує платформи, що задовольняла б зазначені вище вимоги підтримки еволюційності та інтеграційності розглядів. Методологічну основу платформи складають інтуїтивні уявлення про ІТД і рішення ІТ-задач, зібрані у систему загальних та спеціальних принципів. Але перед їх викладенням видається доцільним розглянути ІТД і, зокрема, вирішення ІТ-задач під найбільш загальним кутом зору, абстрагуючись від специфіки предметних областей.

На самому загальному рівні розгляду, з ІТД, зокрема з розробкою апаратного та програмного забезпечень (АтаПЗ), асоційовані принаймні три основних аспекти: прагматика, семантика та синтаксис (рис. 1.1).

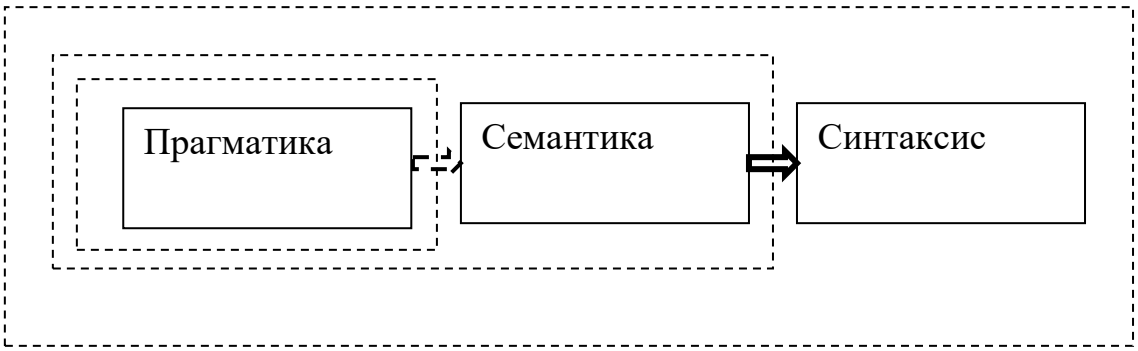


Рис. 1.1 Основні аспекти ІТД

Найвагомішим серед них, безперечно, є прагматика. Саме тут закладаються основні риси як процесу вирішення задачі, так і очікуваного результату. Саме прагматика визначає мету, умови та загальні вимоги до вирішення задачі. Семантика є другим за значущістю аспектом ІТД. Тут

досліджуються питання вибору прагматико-обумовлених засобів вирішення задач і їх застосування. Змістовно кажучи, семантичний аспект пов’язаний із суттю вирішуваної задачі, її прагматико-обумовленою природою та структурою відношень у множині сенсів або значень, представленням «складних» сутей через більш «прості». Таким чином, семантика задачі націлена на представлення вирішення задачі адекватними до прагматичних вимог засобами та у термінах, що презентують відповідні прагматико-обумовлені змісти. Такий змістовий підхід до розгляду задачі забезпечує головне – адекватне розуміння вирішення будь-якої задачі як взаємодоповнення процесу вирішення та його наслідку, зокрема, результату. Це є запорукою реального управління якістю отримуваних рішень, їхньою гарантоздатністю, ефективністю процесів вирішення ІТ-задач і збереження вкладення інвестицій у ІТД. Насамкінець, синтаксичний аспект вирішення ІТ-задач пов’язаний із правильністю побудови форм нотації отриманих прагматико-семантичних рішень, їхньою внутрішньою структурою і т.п.

Наведеного, навіть дуже поверхневого, аналізу основних аспектів ІТД вже достатньо для того, щоб зробити обґрунтований висновок про те, що адекватне розуміння ІТД вимагає залучення до розглядів всіх його трьох аспектів в їх взаємодоповненні. Тому усвідомлення природи цієї взаємодоповнюваності є дуже важливим для розуміння ІТД. Навіть використовуючи наведені вище змістовні міркування, легко прийти до розуміння, з одного боку, природної підпорядкованості аспектів як форми їх взаємодоповнення, а з іншого – необхідності забезпечення їх відносної автономізації у цьому взаємодоповненні. У більш детальному розумінні, семантичний аспект є підпорядкованим прагматичному аспекту, а синтаксичний є похідним від семантичного аспекту. Це безпосередньо випливає зі сказаного про них. Водночас така підпорядкованість не виключає можливості їх відносної автономізації. А саме, прагматика вирішення ІТД задачі може бути відокремлена від її семантики, а семантика – від синтаксису. Таке розуміння взаємодоповнення прагматики, семантики та синтаксису приводить до усвідомлення ІТП як послідовного

сходження від прагматики задачі до її семантики, а від неї – до синтаксису. Причому природа сходжень принципово різна. У випадку «прагматика-семантика» говориться про підпорядкованість у широкому розумінні, що не виключає ані участі суб'єкта у такому сходженні, ані випадку, коли семантика є об'єктивно похідною від прагматики. Щодо випадку «семантика-синтаксис», то тут підпорядкованість суттєво більш конкретна – синтаксис розглядається суто як похідний від семантики у тому сенсі, що перехід є об'єктивізованим.

Зрозуміло, що таке розуміння ІТД за необхідністю вимагає інструментів, які б реально підтримували таке взаємодоповнення прагматики, семантики та синтаксису. Це означає, що володіли б засобами прагматичних, семантичних і синтаксичних розглядів у їх взаємодоповненні та водночас мали б можливість їхньої автономізації у викладеному вище сенсі.

Наведена точка зору є природною не тільки для ІТД, а й стосується рівною мірою будь-якої творчої діяльності. Тут отримано багато результатів різної глибини та значущості. Напевно, центральним серед них є виокремлення системи методологічних принципів, що мають бути покладені в основу будь-якої діяльності (див., наприклад, [11]та бібліографію). Це принципи

- структурності;
- підпорядкованості;
- обумовленості;
- відокремлюваності;

Розглянуті вище підпорядкованість і відокремлюваність основних аспектів ІТД як основа їх взаємодоповнюваності фактично є проекцією відповідних *принципів підпорядкованості* та *відокремлюваності* на галузь ІТД. *Принцип структурності* в контексті ІТД, говорить про те, що *ІТ-рішення – це структурна сутність*. Цим обумовлюється важливість залучення до розглядів поряд із ІТ-рішеннями й властивих їм структур. *Принцип обумовленості* є похідним від відомого епістемологічного закону про примат генетичних структур і стверджує, що *структури ІТ-рішень обумовлені їх генетичними структурами*. Змістовна

суть цього принципу полягає у тому, що властивості ІТ-рішення як структурної сутності обумовлюються його генезисом.

Значимість цієї системи принципів полягає у тому, що вони не тільки чітко окреслюють роль взаємодоповнення основних аспектів ІТД як системоутворюючої основи розуміння ІТД. Як випливає з зазначеного, дотримання принципів імплементує розуміння ІТД як еволюційного взаємодоповнення відповідного процесу і його наслідку. Нажаль, у багатьох випадках стикаємося з їх порушенням у ІТД, розробці АтаПЗ і т.п. Одним із проявів цього є акцентування уваги на результатах розробки АтаПЗ, без урахування специфіки процесів їх створення. Крім того, до них слід віднести штучне обмеження засобів генезису результатів розробки АтаПЗ, надмірну обтяженість розглядів несуттєвими деталями, фактичне домінування синтаксичного аспекту над семантичним і прагматичним аспектами і т.п.

Головним мотивом явно згадати про них були ті суттєві втрати, які є прямим наслідком ігнорування цих принципів. Досвід показує [12] [13], що втрати, які є причинами актуальної замкненості традиційних підходів до ІТД, не тільки не мінімізуються, але й, мабуть, не завжди усвідомлюються. Це випливає навіть із назв багатьох підходів, стилів і методів вирішення ІТ, зокрема програмістських, задач. Короткий змістовний аналіз згаданих втрат і неадекватностей, властивих основним парадигмам розробки ПЗ, як виду ІТД, є дуже важливим для прагматико-обумовленої побудови композиційної платформи моделювання ІТД.

1.1.1 Функціональна платформа

Тут найвищий пріоритет віддається поняттю функції, а саме – функціоналу як одному з видів функції. Щодо композиції, то вона як поняття взагалі відсутня та представлена у функціональному підході (ФП) лише на рівні малопоказових прикладів, які не узгоджуються ні зі структурою функцій, ні тим паче зі структурами даних. Це призвело до дуже жорсткої структури даних –

послідовності послідовностей, яка стала «прокрустовим ложем» для масивів, записів та інших важливих структур даних реальної розробки ПЗ.

Подібні труднощі добре відомі. На них все частіше вказують як творці ФП, так і його активні послідовники та пропагандисти (див., наприклад, [14] [15] [16] [17]) При цьому, однак, звертається увага на каталогізацію цих труднощів, а не на розкриття джерел їх виникнення. Головним їхнім джерелом є зміщення акцентів із композицій на функціональні структури, задані на структурах даних, які абсолютно не адекватні логіці розробки ПЗ, хоча і досить вдало відображають істотні специфічні риси спеціальних розділів дескриптування [18], зокрема програмування. Це не суперечить очевидній екстенсіальній повноті цих структур, але свідчить про їх інтенсіальну (змістовну) обмеженість. Проявляється вона в тому, що послідовності послідовностей як структури даних «нав'язують» обтяжливо зайву, не пов'язану з сутністю розв'язуваної задачі конкретизацію у представленні оброблюваних об'єктів. Тобто специфіка таких структур є «баластом», що утруднює вирішення задачі. Інакше кажучи, справедливо наступне положення.

Положення 1. Функціональна платформа замкнута в актуальності функцій.

У цьому сенсі ФП не є адекватним збагаченням розробки ПЗ. Це випливає з огляду на зазначену вище відкритість відповідного виду ІТД. Загалом така неадекватність не обов’язково є наслідком тільки замкнутості платформи в актуальності. Вказана замкнутість є лише достатньою, але не необхідною умовою. Не виключаються й інші чинники неадекватності, яких може бути дуже багато.

1.1.2 Об'єктно-орієнтована платформа

В об'єктно-орієнтованому підході (ООП) на перше місце висувається поняття об'єкта. При цьому воно розглядається на гранично високому рівні абстракції. Структури даних, функцій, композицій і декомпозицій трактуються як окремі випадки об'єктів, які явно не індивідуалізовані в універсумі об'єктів на належному рівні конкретики. Без сумніву, таке інтегроване тлумачення є

перевагою підходу. Але, як часто буває, недоліки є продовженнями переваг. Така тотальна увага до всіх об'єктів «оптом» залишила поза межами розгляду питання про пріоритетність у трійці понять – «композиція», «функція», «дані», а й позбавило належної уваги до кожного з них окремо, вивело поза межі розглядів питання про причинно-наслідкові зв’язки процесів породження об’єктів із об’єктами породження. Сформулюємо сказане у вигляді положення.

Положення 2. ООП замкнута в актуальності об’єкту.

1.1.3 Логічна платформа

Тут акцент робиться на засобах логіки у розробці ПЗ. Це зближує його з логіко-предметним підходом і логікою розробки зокрема. Однак ці засоби у нинішньому стані обмежені можливостями чисто декларативних логік, крім того, лише окремих їхніх розділів, типу логік першого порядку. А ці можливості, як відомо, є дуже обмеженими. З результатів, викладених вище, випливає, що реальне програмування характеризується передусім дескриптивними логіками [18] [19] [20] [21] [22] [23] [24], що базуються на функціоналах вищих типів, індукованих композиціями як логічними засобами генезису рішень. Тому логічна платформа (ЛП), декларативні логіки якої не обумовлені дескриптивними логіками як генетичними структурами, в силу принципу обумовленості лише за формою (назвою), а не по суті близька логіці розробки ПЗ.

Конкретні труднощі, з якими зіткнулася сучасна ЛП, добре відомі. Причини багатьох із них розкриті класиком підходу Дж. Робінсоном у його визначній роботі [25]. Це відсутність засобів логік вищих порядків, ігнорування відношень і функціоналів вищих типів, неадекватність функціональних структур і структур даних і т.п. У зв'язку з гостротою, яка притаманна таким негативним явищам у реальних програмах, посилились намагання провадити у життя ідею – об'єднати логічне програмування з функціональним. Одним із перших помітних результатів, отриманих на цьому шляху, став Логліср [25], що являє собою реалізацію логічної парадигми у програмотворенні в межах Ліспа.

Незважаючи на великі надії, які пов'язують із подібними інтеграціями, вони навряд чи приведуть до якісно нових зрушень у програмуванні і ІТД взагалі. Справа в тому, що ті причини, які прагнуть усунути, по суті є не причинами, а лише наслідками головної з першопричини. А саме, відриву (декларативного) логічного програмування від (дескриптивної) логіки програмування, що проявляється конкретно в ігноруванні композицій, які лежать в основі цих логік і являють собою адекватні експлікації засобів генезису рішень. Таким чином, справедливе

Положення 3. ЛП замкнута в актуальності декларативної логіки.

1.1.4 Структурна платформа

Структурний або, як ще кажуть, систематичний підхід (СП) акцентує увагу на структурі ПЗ, індукованих систематичними (структурованими) прийомами їхньої побудови. Цим самим звертається увага не тільки і навіть не стільки на самі ПЗ, програми, скільки на процеси їхньої побудови. З огляду на принципи структурності та обумовленості цілком все це узгоджується з концептуальним базисом АПС у цілому і композиційною логікою розробки зокрема. Щодо конкретного змісту структурного програмування, то воно розвивалося головним чином навколо найважливіших керуючих структур типу послідовного виконання, галуження та циклування. На цьому шляху, як відомо, були отримані вагомі результати, зокрема в області програмної інженерії. Однак і тут не перейшли від використання (часто дуже ефективного і ювелірно витонченого) конкретних керуючих структур до розгляду адекватних керуючих структур як поняття (по суті поняття композиції) з точно визначеними відповідними семантикою та синтаксисом останніх. Хоча одержувані при цьому втрати полягають навіть не у тому, що з повним виключенням оператора go to з класичного структурного програмування, як переконливо обґрунтував Д.Кнут у своїй відомій праці [26], “вихлюпнули з водою і дитину”, а в тому, що цим не було здійснено поштовху для якісно нового етапу розвитку так блискуче початого систематичного підходу в розробці та ІТД.

Зі сказаного випливає

Положення 4. СП замкнута в актуальності статичної структури.

1.1.5 Модульна платформа

Модульний підхід зосереджує увагу на понятті модуля, тобто знов-таки — фактично на понятті функції, а не композиції. Водночас очевидно, що визначальним тут є міжмодульний інтерфейс, а не модуль. Але міжмодульний інтерфейс і є композицією. Тому логічно, що тут потрібно було б робити наголос не на чисто зовнішніх по відношенню до суті формах паспортизації міжмодульних інтерфейсів, а на розкритті їх змісту, тобто семантики (композицій).

Зміщення акцентів у тріаді «композиція, функція, дане» не обмежується абсолютизацією поняття функції. Іноді така абсолютизація переноситься навіть на поняття даного. Так, наприклад, у методі Джексона (програмування від даних) пріоритет віддається поняттю даного з витікаючими звідси перевагами та недоліками. При цьому головний недолік не в недостатньо широкому трактуванні даних, як це прийнято розглядати в підходах, що базуються на так званих абстрактних типах даних [27]. Як випливає з отриманих результатів, причина полягає у тому, що не враховується не тільки прагматико-цільова пріоритетність понять згаданої тріади, а й самі поняття і її складові залишаються поза ґрунтовним розглядом. Тому абстрактні типи даних у нинішньому їх стані мало чим могли б тут допомогти, оскільки останнє зауваження повною мірою відноситься й до них. Підсумуємо сказане у вигляді наступного положення.

Положення 5. Модульна платформа замкнута в актуальності модуля.

1.1.6 Денотаційна платформа

Денотаційний метод заснований на загальному методі нерухомих точок (див., наприклад, [28] [29]). Акцент робиться на цьому загальному методі як засобі отримання явних визначень за спеціальними класами визначень через відношення, зокрема, визначень у вигляді систем рекурсивних рівнянь. Інакше

кажучи, зосереджується увага на засобі, що забезпечує можливість зведення проблеми явних (конотативних, процедурних) визначень до проблеми неявних (денотативних) визначень (звідси походить назва «денотаційний метод»).

З огляду на те, що денотативні визначення об'єктів, зокрема даних, функцій і композицій, часто кращі, ніж конотативні, денотаційний метод широко використовується в розробці як програмного забезпечення, так і в інших галузях знань. Однак, незважаючи на широту використання, денотаційний метод як метод нерухомої точки все ж є метазасобом по відношенню до засобів розробки – композицій, функцій і даних. І його абсолютно недостатньо, щоб із нього безпосередньо виводити засоби розробки, програмування, моделювання і т.п. У найліпшому разі денотаційна платформа (ДП) може тут виступати як допоміжний компонент за умови, якщо ДП як метазасіб буде відповідно до принципів структурності, обумовленості, підпорядкованості та відокремлюваності належним чином узгоджено з конотативно-денотативною генетичною структурою пріоритетності понять згаданої вище тріади. Але тут поки що панує недостатність розуміння, яка істотно гальмує розвиток теорії і практики розробки. Отже, сказане підтверджує наступний висновок.

Положення 6. Денотаційна платформа замкнута в актуальності денотації[1].

Проведений аналіз основних парадигм розробки ПЗ свідчить про те, що, всі вони дотримуються виділеної системи принципів лише номінально. І тому згадані вище недоліки властиві їм у більшою або меншою мірою. Зважаючи на фундаментальність причин їх виникнення, ці недоліки не можуть бути природним чином знівельовані чи усунуті. Змістовно кажучи, вони є принциповою складовою цих підходів! З огляду на це приходимо до важливого предметного наслідку.

Наслідок. Кожне взаємне доповнення функціональної, об'єктної, логічної, структурної, модульної, денотаційної платформ наслідує їхні актуальні замкнутості.

[1] Зрозуміло, що подібний висновок можна зробити і відносно конотативної платформи.

Екстраполюючи результати, можна стверджувати, що зазначене свідчить про неможливість у рамках традиційних підходів вирішувати задачі адекватного залучення ІТП, зокрема, програмування, до розглядів. Тому необхідним є створення нового інтеграційного підходу, що буде зберігати позитиви існуючих підходів і забезпечувати адекватне залучення як ІТП та його наслідку, так і їх взаємодоповнення до розглядів.

Усвідомлення цих проблем стало основним мотивом до розвитку композиційного підходу до вивчення ІТД [30] [12] [31] [32] [33] [34] [35] і розвитку на його основі дескриптології – науки про ІТД [18] [11] [34] [35] [21] [22] [23] [24], яка включає три складові частини:

• методологію дослідження дескрипцій і дескриптування;

• теорію дескриптивних середовищ;

• технологію дескриптування.

Дана робота спрямована на подальший розвиток дескриптології у напрямку ІТД як носія спеціальних видів дескрипцій і дескриптування. До таких видів, зокрема, відносяться АтаПЗ та розробка АтаПЗ.

У наступному розділі уявлення про ІТ-рішення, відповідний процес та їх взаємодію збагачено саме за рахунок втягнення у реальні розгляди трьох основних аспектів ІТД: прагматичного, семантичного, синтаксичного. Розгляд їх відносно автономізований, що диктується прагматикою ефективності ІТД. Водночас зв’язки між ними, грають ключову роль у дослідженні. Сказане відображається у ряді принципових положень ІТД, що є предметними конкретизаціями відомих методологічних принципів: структурності, підпорядкованості, обумовленості, відокремлюваності.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ЗАСАДИ АПС

2.1 Збагачення розуміння ІТД

2.1.1 Сутесутнісна природа розуміння ІТД

Основу будь-якої, зокрема і ІТД як мотивованого процесу досягнення мети, складає її розуміння. Воно є джерелом особливостей такої діяльності, її слабих і сильних сторін, переваг та недоліків. Саме тому, вище зазначені задачі можуть бути ефективно вирішенні тільки на основі усвідомлення природи ІТД. З огляду на її надзвичайну відкритість таке усвідомлення бачиться на шляху поступових прагматико-обумовлених збагачень уявлень про природу її розуміння. Точкою відправлення тут, напевно, є головна та стала особливість розуміння ІТД – притаманна йому релятивність, обумовлена здатністю його до еволюційного розвитку. Проявляється це, наприклад, у тому, що розуміння предмету ІТД еволюціонує разом із прагматико-обумовленим збагаченням уявлень про нього. Така релятивність розуміння, а ІТД дозволяє залучати до розглядів, зокрема, її крайні прояви у якості як процесу, так і його результату. Не зважаючи на те, що крайні трактовки теж мають право на існування, адекватним є погляд на нього з точки зору прагматико-обумовленого компромісу між цими крайнощами. Зміст його в першому наближенні, полягає в тому, що розуміння в загальному контексті, зокрема і розуміння ІТД, пов'язується не стільки з результатом (наслідком) її розуміння та навіть не з її причиною (передумовою), скільки з їх взаємодією. Така позиція обумовлюється загальнозначущою роллю причинно-наслідкового зв'язку та інтуїтивним баченням як власне розуміння в цілому, так і розуміння ІТД як його предмету[2]. Тобто, згаданий компроміс у першому наближені полягає в тому, що розуміння розглядається як діяльність, можливий наслідок якої *обумовлено* відповідною передумовою. У якості передумови (причини) тут може виступати будь-який об’єкт ІТД як носій розуміння.

[2] Розуміння в цілому розглядається тут як засіб пізнання, а розуміння ІТД – як предмет пізнання.

Таким чином, з огляду на вищевикладене, обґрунтованим є наступне розуміння ІТД:

ІТД – суть обумовлення об'єкту ІТД.

Наведена експлікація реально збагачує інтуїтивне уявлення про розуміння ІТД. Вона експлікативно зводить подальшу інтенсіалізацію розуміння ІТД до інтенсіонального збагачення вже *обумовлення об'єкту ІТД.* Дуже важливо при цьому не втратити нічого значущого в процесі збагачення. У якості платформи збагачення тут обрано дескриптологію [32] [12] [34] [35] [22] та розвинуту на цій основі теорію дескриптивних середовищ [19] [20] [22] [36]. Для цього, як мінімум, і *об'єкт ІТД,* і *обумовлення* повинні трактуватися максимально широко. Стосовно першого, слід зазначити, що його можна трактувати як те, що охоплює, як реальне, так і ідеальне, як загальне, так і окреме, як абстрактне, так і конкретне, як статичне, так і динамічне, як об'єкти, так і суб'єкти і т.д. Тобто, все те, що може бути. Враховуючи важливе значення введеної експлікації, домовимось це трактування об'єкту ІТД назвати *сутністю* [19] [20] [22] [36].

У контексті розуміння ІТД визначальне значення займають не просто сутності взагалі, а ті, що обумовлюються сутностями,

суть = сутність, що обумовлюється сутністю [19] [20] [22] [36].

Наведена експлікація суті, звичайно, не є визначенням, але має визначальне значення. Вона є прагматико-обумовленим шаблоном подальших експлікацій, які потенційно можливі в еволюційному розвитку розуміння. Проявляється це, перш за все, в реальній, а не номінальній можливості об'єктивування участі суб'єкта в такому розвитку. Незважаючи на це, зрозуміло, що дана трактовка вкрай загальна, об'ємна, а значить – малозмістовна. Тому вона потребує подальшого збагачення. Всі подальші побудови та збагачення, як сутності, суті, так і їх причинно-наслідкового зв'язку присвячені, по великому рахунку, розгляду ІТД з дескриптивної точки зору – точки зору, що розглядає ІТД як сутесутнісну [23] взаємодію ІТП та його наслідку.

Таким чином, у контексті прагматики розробки апаратного та програмного забезпечення наведені експлікації сутності й суті хоча й важливі, але не стільки самі по собі, скільки у їхньому причинно-наслідковому зв'язку, що пов’язує сутність як передумову обумовлення із суттю – обумовленням передумови. Ця залежність становить кістяк інтуїтивного бачення *розуміння ІТД як розкриття суті сутності*:

Сутесутнісна парадигма. Розуміння ІТД експлікативно зводиться до розкриття суті сутності.

Значимість викритого зв'язку обумовлює реальність його експлікації у якості відношення на сутностях і сутях як найважливішого виду причинно-наслідкового зв'язку, який природно назвати *сутесутнісним відношенням* (ССВ) [36]. Останнє є бінарним відношенням, яке з огляду на проведені експлікації цілком природно розглядати як рефлексивне, антисиметричне і транзитивне, тобто як частковий порядок. З усього вищенаведеного випливає

Сутесутнісне положення. ІТД – суть рефлексивно-транзитивне замикання ССВ.

Такий розгляд реально збагачує введену вище експлікацію розуміння й експлікує собою генезис понять у їх еволюційному розвитку як наслідків розуміння. Змістовно кажучи, так введене відношення являє собою загальнозначущу «схему розуміння» й основний інструмент еволюційного збагачення понятійної системи. Остання еволюціонує шляхом розгортки сутесутнісного відношення як «руху по частковому порядку». Що ж до обумовлення, то широта його трактування носить інтерсуб’єктивний характер, тобто є загальною настільки, що навіть обезумовлення трактується як обумовлення безумовою (негацією обумовлення). Відкритість схеми як запорука її загальнозначущості (в цьому сенсі, логічності) забезпечується зазначеною інтерсуб'єктивністю як «обумовленості», так і сутності загалом.

Сутність, суть [37] і сутесутнісне відношення складають кістяк понятійної системи адаптивних процесональних середовищ (АПС), що підтримує релятивну природу ІТД. Його розвиток пов'язаний із подальшими, як загальнозначущими,

так і предметними збагаченнями. Причому домінантою тут є ССВ, що уособлює собою взаємодоповнення сутності та суті у контексті розуміння. Збагачення здійснюються у вигляді прагматико-обумовлених застосувань зазначеного інструменту разом із прагматико-обумовленим замиканням відкритості обумовленості.

2.1.2 Композитосутнісне збагачення розуміння ІТД

Головним мотивом здійснюваних збагачень є поступове, як безпосереднє, так і опосередковане роз'яснення природи ССВ. З огляду на причинно-наслідкову природу сутесутнісного відношення воно тісно пов'язане з обумовленням. Тому його подальше реальне збагачення бачиться в розвитку зв'язку причини та наслідку як основних атрибутів будь-якої діяльності. Головною особливістю будь-якої діяльності, зокрема й інформаційно-технологічної, є її суб'єктно-об'єктна природа. Остання ж тісно пов'язана з активною та пасивною формами здійснення діяльності. Змістовно кажучи, активна форма уособлює собою відповідний процес, а пасивна - пов’язана з його наслідком. Взаємодоповнення цих форм є ключовою характеристичною особливістю породжуваного діяльністю причинно-наслідкового, а в разі обумовлення – сутесутнісного відношення – відношення, що підтримує перехід від сутності до суті. Воно ж є основним плацдармом подальшого реального збагачення ССВ. Продемонструємо це у вигляді ланцюга сутесутнісних збагачень.

Відповідно до наведеної експлікації суть трактується як сутність, яка обумовлюється «в принципі». Тобто обумовлюваність втягується тут на рівні ознаки, характеристики сутності. Цього для збагачення звичайно ж недостатньо. Відштовхуючись від експлікацій сутності, суті та суб’єктно-об'єктної природи ССВ, коректно описати механізм «набуття» цієї ознаки. Ключове місце тут займають не просто суті, як сутності, що обумовлюються сутностями – суб’єктами обумовлення, а ті з них, що самі обумовлюють таких суб’єктів. Вони є, в цьому сенсі, джерелом прагматико-обумовленого концептуально-єдиного

бачення взаємодоповнення сутності та суті в ССВ як переходів від сутності до суті і навпаки – від суті до сутності. Тому суть, що обумовлює сутність таким концептуальним баченням, природно назвати концептом, а сутність, яка є об'єктом концептного обумовлення – монадою (від грец. monados – одиниця) [37]. Таким чином, приходимо до експлікації важливого виду ССВ як концептно-монадного збагачення суб'єктно-об'єктного (активно-пасивного) взаємодоповнення.

$$\begin{cases} \textit{концепт} = \textit{суть, що обумовлює сутність}; \\ \textit{монада} = \textit{сутність, що обумовлюється концептом}; \end{cases}$$

Концепт і монада звичайно збагачують ССВ, індивідуалізуючи найважливіший вид діяльності – концептування. Однак у прагматиці, орієнтованій на розкриття природи ІТД, цього збагачення недостатньо. Пов'язано це з надмірно широким трактуванням концепту як суб'єкта обумовлення. Вище було зазначено, наскільки важлива для ІТД процесійна (динамічна) точка зору на ІТД та її результати. Однак ефективно підтримати динаміку, так само як і статику, і, тим більше, їх взаємодоповнення в рамках концептування не видається можливим. Адже концепт не передбачає індивідуалізації видів обумовлення. Це випливає з того, що передумова «концептуального» обумовлення – сутність, є максимально загальною, а значить малозмістовною. Тому концепт не може бути покладений в основу статико-динамічної точки зору на ІТД. Необхідним стає подальше збагачення концептування, основу якого складає концепт, як суб'єкт діяльності.

Для концептування як виду діяльності характерні важливі в контексті розуміння ІТД аспекти. Це обумовлено біфуркативністю природи обумовлення як і діяльності в цілому, якій притаманні статичний і динамічний види. Різниця між ними принципова. Статична діяльність обумовлена станом деякої сутності, відношеннями між сутностями і т.п., а динамічна - безпосередньо пов'язана з деяким дією, актом, акцією. Неврахування статико-динамічної природи концептування невиправдано збіднює розгляд монад. Тому необхідна індивідуалізація динамічного та статичного видів концептування.

Динамічне концептування, яке випливає з принципу процесійності, є основоположним видом концептування. Тому слід розпочати збагачення саме звідси. Основу збагачення складають акт і акція. Їх експлікація наведена нижче [18].

$$\begin{cases} акт = концепт, що\ монадно\ обумовлює\ монаду \\ акція = монада, що\ обумовлюється\ актом \end{cases}$$

Змістовно кажучи, у введеній експлікації акт - це не просто концепт, а такий, у якому, по-перше, в якості передумови обумовлення виступає вже не максимально загальна сутність, а суттєво більш змістовна монада. По-друге, сам спосіб обумовлення теж розглядається більш змістовно. Він, у свою чергу, є монадою. Значить, як мінімум, коректно ставити питання про відповідний йому концепт, притаманне йому концептування і т.п. Таким чином, акту притаманна набагато змістовніша динаміка в порівнянні з концептом. Адже він не просто обумовлює, а монадно обумовлює, до того ж, не просто сутність, а монаду. У зв’язку з цим, акція вже не просто монада, а монада, що обумовлюється актом, природа якого за необхідністю є суб’єктивною, в тому сенсі, що тут є прагматико-обумовлена можливість коректно ставити питання про суб’єкта монадного обумовлення.

Експлікація акціонування, звичайно ж, допускає подальші безпосередні нетривіальні збагачення. Це обумовлено відкритістю самих акту та акції. Однак збагачення завжди повинні спиратися на реальні уявлення про збагачувані об'єкти. Сьогодні ж можна констатувати, що зведення динамічного концептування до акціонування є лейбніцевим [38] [18], а значить – експлікативним.

Глибинною основою всіх побудов, представлених у роботі є інтеграція динамічних і статичних аспектів середовища розуміння. Вона проявляється в тому, що прагматико-обумовленою домінантою в ній є примат динаміки над статикою. З цього випливає, що об’єктивно лише він реально, а не номінально

може підтримувати концептуальну єдність еволюційного розвитку розуміння та, зокрема, його середовища. Для того, щоб природним чином використати цю можливість слід збагатити відношення між актом і монадою. Експлікуємо це так.

$$\begin{cases} \textit{поліспект} = \textit{акт, що багатоадно обумовлює монаду}; \\ \textit{поліада} = \textit{монада, що обумовлюється поліспектом}. \end{cases}$$

Принциповим тут є те, що інтерсуб'єктивною основою цієї експлікації є акт. Але не просто акт, а такий, що багатоадно обумовлює монаду. Звідси, зокрема, випливає, що багатоадність носить не статичну, а динамічну природу. Цим явно акцентується увага не на складові монади, а на засобі їх «збирання». Це не обов'язково канторове збирання ([39], стор.33). Апріорі природа цих збирань нічим не обмежена. Вона інкапсулюється (приховується) з можливістю, як у часі, так і в просторі у разі необхідності розкрити її. Все це природним чином підтримує адекватність не тільки засобу побудови – поліспекту, а і його результату – поліади.

Наведена система збагачує концептування новим його видом – збиранням, суб'єктом якого є поліспект як носій «багатості»[3]. Значимість його полягає, перш за все, у тому, що збирання, будучи адекватним інструментом збагачення традиційних розглядів по лінії «ціле – складене», одночасно володіє серйозними резервами розвитку й у ІТ-проблематиці. Для реалізації цих резервів серед усього розмаїття аспектів збирання доцільно виділити важливі для ІТД. Видається, що слід звернути увагу на аспекти, обумовлені тим, що з будь-якою досліджуваною сутністю асоційовані як внутрішні, так і зовнішні її характеристики, ознаки, властивості, зв'язки. Залучення і тих, і інших у розгляд може бути обумовлено відповідною прагматикою. Тому поліади як об'єкти збирання повинні природним чином забезпечувати можливості таких залучень.

[3] Багатість – від слова багато. Можна було б, звичайно, використати слово «множинність», якби не його безпосередній зв'язок із множиною. Зміст слова «багатість» у тому, що він є родом для таких його видів, як, наприклад, множиність, агрегатність, кортежність і т.п. варіації сприйняття багатьох речей як єдиного цілого.

Точка зору, в якій внутрішні властивості сутностей є абсолютною домінантою розгляду, через призму якої вбачаються вже всі інші їх характеристики, пройшла серйозну перевірку часом, а отримані результати довели її спроможність, перш за все, у традиційній прагматиці. У прагматиці ж ІТД все більш очевидним стає неадекватність абсолютизації лише внутрішнього бачення досліджуваних сутностей. Тому розгляд внутрішньої природи сутності у взаємозв'язку як із її зовнішньою природою, так і з взаємодоповненням цих природ є найважливішою особливістю розгляду. Водночас природно, що первинним для неї є врахування саме внутрішньої, змістовної або інтроспектної (від лат. introspecto – дивлюся всередину) обумовленості втягуваних у розгляд поліад. Тому суб'єкти такого обумовлення будемо називати інтроспектами, а його об'єкти, віддаючи данину традиціям – множинами, маючи на увазі, що даний вид поліад є дещо багатшим за будь-який його традиційний аналог, у більшості за рахунок суб'єктної відкритості роду інтроспектів [22].

$$\begin{cases} \textit{інтроспект} = \textit{поліспект, що інтроспектно обумовлює поліаду;} \\ \textit{множина} = \textit{поліада, що обумовлюється інтроспектом.} \end{cases}$$

Незважаючи на все багатство множин як видів поліад , їх, за зазначеними вище причинам, все ще недостатньо для забезпечення адекватних ІТ-прагматиці та розглядів сутностей. Необхідні нові збагачення, що дозволяють адекватно розширити кругозір, обмежений поки лише внутрішніми властивостями сутностей. Перш за все, за рахунок залучення в розгляд їх окремих важливих зовнішніх ознак. Одна з важливих властивостей множин обумовлена тим, що їх елементи можуть бути зовні пов'язані один із одним. Все різноманіття таких зв'язків є потенційно відкритим. Однак особливе місце серед них займають т.зв. лінійні зв'язки, обумовлені лінійними порядками. Адже відповідно до леми Цорна [39] [40] будь-який, як завгодно складний частково-упорядкований зв'язок (ланцюг) може бути зведений до лінійно-впорядкованого ланцюга (зв'язку). У

цьому сенсі лінійні ланцюги є надпотужним інструментом (екстраінструментом) представлення як завгодно складних зв'язків елементів множин. Тому сутності, що лінійно обумовлюють поліади, назвемо екстраспектами, а об'єкти таких обумовлень – екстрамножинами [22]. Видами екстрамножин є, наприклад, кортежі та мультимножини. Таким чином, маємо

$$\begin{cases} \textit{екстраспект} = \textit{поліспект, що екстраспектно обумовлює поліаду}; \\ \textit{екстрамножина} = \textit{поліада, що обумовлюється екстраспектом}. \end{cases}$$

Експліковані види сутесутнісних динамічних та статичних відношень, звичайно ж, допускають подальші безпосередні нетривіальні збагачення по лініям динаміки та статики. Це обумовлено їх вихідною відкритістю. Однак, не зважаючи на принципову значимість таких збагачень для розбудови середовища розуміння в цілому, їх розгляд у повному обсязі вивів би нас далеко за рамки прагматики цієї роботи. Тому не будемо зупинятись на їх всеохоплюючому розгляді, а акцентуємо увагу лише на ті з них, що мають достатню прагматико-обумовлену мотивацію.

З огляду на зазначене подальші збагачення, як сутесутнісного відношення, так і основних його складових – сутності і суті, доцільно пов'язати з взаємодоповненням динаміки та статики як власне концептування, так і його видів. У цьому напрямку досить прозоро експлікується ряд найважливіших, динаміко-статичних збагачень, які, з огляду на популярність їх змістовних прообразів наведемо без їх попереднього докладного обґрунтування.

$$\begin{cases} \textit{функутор} = \textit{інтроспект, що акційно обумовлює множину}; \\ \textit{функція} = \textit{множина, що обумовлюється функтором}. \end{cases}$$

$$\begin{cases} \textit{операкт} = \textit{акт, що кортежно обумовлює акцію}; \\ \textit{операція} = \textit{акція, що обумовлюється операктом}. \end{cases}$$

$$\begin{cases} \textit{компакт} = \textit{операкт, що генетично обумовлює операцію}; \\ \textit{композиція} = \textit{операція, що обумовлюється компактом}. \end{cases}$$

$$\begin{cases} актоїд = акт, що\ композиційно\ обумовлює\ акцію; \\ оператор = акція, що\ обумовлюється\ актоїдом. \end{cases}$$

Значущим у останніх експлікаціях є те, що всі вони – результати індивідуалізації важливих видів динаміко-статичних обумовлень. Адже, як вже зазначалося, взаємодоповнення динаміки та статики – ключ до адекватного прагматиці розуміння ІТД. Стосовно розуміння функції, то тут ми виходимо з того, що хоча вона є безумовно важливою у контексті обумовлюючої її функційної ситуації, що розглядається. Така ситуація, звана надалі *функтором*, виникає, наприклад, коли певна дія розуміється як прояв певної закономірності, що трактується як деякий інструмент реагування на ту чи іншу передумову, що виникає в реальності. Фактично функтор є видом інтроспектів, функційність – акційним видом однозначної інтроспектності, а функція – видом множини – множиною акцій. Наступні експлікації збагачують акційність, індивідуалізуючи значущі як для традиційної, так і для ІТ-прагматики, види роду сутностей. Коротко, *кортежність* – обумовлює акт кортежною природою передумов акцій; *генетичність* – вид кортежності, де будь-яка акція операції обумовлена тим, що можливий її наслідок «споріднений» зі складовими передумов самої операції; *композиційність* – обумовлює кортежність операції умовою її адекватності прагматиці [11]. Так уведені розуміння функції, операції, композиції та оператора природним чином обумовлюють їх відкритість. Адже, наприклад, композиція спирається на генетичність компакту, функція - на акційність функтора, а операція – на кортежність операкта. Саме це є запорукою адекватності їх розуміння. Адже як генетичність і композиційність, так і адекватність, у першу чергу, вочевидь носить суб'єктивну, прагматико-орієнтовану природу. Саме цим забезпечують сутесутнісну релятивність і реальність (відкритість «по суб’єкту») розгляду. З одного боку, розгляд за потреби замкнено в композиційній логіці (КЛ), основу якої складає тріада

<композиція, сутність, композитосутнісне відношення>, а з іншого – відкрито з огляду на відкритість як композиції, так і «похідного» від неї оператора. Насамкінець зазначимо, що отримані види сутесутнісних, зокрема композитосутнісних відношень (КСВ), цілком відкриті для наступних збагачень, що все більше враховують аспекти динаміко-статичного взаємодоповнення між ІТД і її результатом. Детальніше з цим можна ознайомитись, наприклад, у [1] [18] [36] [41] [42] [37].

Отримані натепер збагачення ССВ, такі наприклад, як **ф**ункційно**с**утнісні, **о**перaційно**с**утнісні, **к**омпозито**с**утнісні та **оп**ерато**р**о**с**утнісні відношення (ФСВ, ОСВ, КСВ та ОпрСВ відповідно), звісно, не є остаточними. Кожне зі збагачень потенційно відкрите для будь-яких ефективних продовжень. Однак для цього такі продовження повинні спиратись на ґрунтовну прагматико-обумовлену мотивацію. Запропонований ланцюг сутесутнісних збагачень спирається на фундаментальні результати як класичних і неокласичних досліджень в області онтології [43], феноменології [44] [45], гносеології, епістемології [46] [47] [48], логіки [15] [49], математики [50] [51] [52] та ґрунтованих на них парадигмах ІТД (таких, наприклад, як ФП [14] [15] [16], СП [53] [54] [55] [56] [57] [58], ООП [59], ЛП [17] [25], ДП [60], КП [30] [12] [31] [32] [30] [33] [34] [35] [19] [20]), так і на некласичних дослідження в області дескриптивних середовищ ІТД [20] [21] [22] [23] [24] [61] [62]. Цей ланцюг збагачень значно систематизує отримані результати та підводить під них т.з. загальний сутесутнісний знаменник. Це дає можливість, спираючись на даний фундамент, зробити наступний крок у напрямку адаптивних процесональних середовищ – подивитись на ІТД у контексті композиційної логіки розуміння. Ця точка зору виражена в основоположній тезі.

Теза композитосутнісності. ІТД – суть рефлексиво-транзитивне замикання КСВ.

Значимість цієї тези, перш за все, полягає в тому, що вона прагматико-обумовлено інтегрує в собі всі результати, викладені в роботі як логічного (загальнозначимого), так і предметного (специфічного) характеру.

Розглянуті в розділі ССВ та їх види акцентують увагу на взаємодоповненні динаміки та статики, суб'єкта й об'єкта, ІТД та її наслідку як на видах обумовлення. Зі змістовної точки зору та спираючись на вище викладене, стає зрозумілим, що всі вони є засобами, які підтримують еволюційний розвиток сутей та сутностей у їх взаємодоповненні. Тому подальші кроки збагачення розуміння ІТД за необхідністю пов'язані з більш глибоким проникненням у природу цього розвитку.

2.1.3 Відкрито-замкнені аспекти ІТД

В [18] [42] [41] [19] [20] [21] [22] [23] наведене докладне обґрунтування адекватності відкрито-замкнутої платформи як універсального інструменту розгляду досліджуваних сутностей у їх еволюційному взаємодоповненні. Тому саме вона покладена в основу прагматико-обумовленого збагачення ІТ-розглядів.

У першому наближенні її суть зводиться до пошуку прагматико-обумовленого компромісу між замкненою та відкритою точками зору на інформаційно-технологічну, зокрема і ІТД. Відомо, що поряд із безперечними перевагами, останні мають принципові недоліки, які все більше домінують. Це робить актуальним формування такої точки зору на інформаційно-технологічні системи (ІТС), яка б по можливості була вільною від таких недоліків, зберігаючи при цьому переваги як замкнутих, так і відкритих точок зору. Коротко будемо називати відкрито-замкнену точку зору на сутності OC-System (Open-Close System).

Першопричина, що лежить в основі згаданого компромісу полягає в тому, що ані замкнуті, ані відкриті системи не підтримують головне у розглядах – ранжування залучених сутностей з урахуванням їх прагматико-обумовленої значущості. Саме тому природно приходимо до висновку про те, що OC-System

повинна підтримувати вищезгадане прагматико-обумовлене ранжування розглядів за ступенем значущості. При цьому різниця в значущості тут не просто декларується, а природно реалізується у вигляді принципово різного використання значимого та другорядного типів абстракції у взаємодоповнюючих розглядах.

Перший утворює прагматико-обумовлене замкнуте ядро, другий - є представником відкритої системи продовжень цього ядра. Це дозволяє з самої загальної точки зору дивитися на ВЗС як на прагматико-залежний біабстрактний зв'язок відносно автономізованих замкнутої (в цьому сенсі загальнозначущої, логічної) і предметної частин системи як принципово різних типів абстракції. Логіко-предметна спрямованість є визначальною в цілому для концепції ОС-System. При цьому як логіка, так і предмет тут розуміються релятивно та можуть бути в свою чергу розглянуті логіко-предметно. Ця відносність дозволяє поряд із відкрито-замкнутими системами говорити про відкрито-замкнені середовища (ВЗСр) як середовища існування ВЗС. Центральне місце у цілісному розгляді будь-якого ВЗСр поряд із його замкненим ядром (логікою середовища, макросередовищем) і відкритою системою можливих предметних продовжень ядра (мікросередовищем) займає індуковане відносною автономізацією макро та мікросередовищ ВЗСр прагматико-обумовлене їх логіко-предметне відношення (ЛПВ). Останнє є бінарним відношенням, яке у зв'язку зі згаданою релятивністю логіки та предмета середовища природно розглядати як рефлексивне, транзитивне й антисиметричне, тобто як частковий порядок.

Це дозволяє змістовно роз'яснити природу ВЗСр як прагматико-обумовленого логіко-предметного відношення над його макро і мікросередовищами. Очевидно, що таке трактування збагачує поняття відкрито-замкненої системи, роблячи можливим дослідження його у рамках ВЗСр. Як було показано в [18] [42], універсальним засобом вивчення сутностей, зокрема і ВЗС є прагматико-обумовлена типізація (ТОП) універсуму сутностей (УС). Вона, як відомо, зводиться до прагматико-мотивованих уведень і виключень абстракцій в УС як несуперечливій логічній абстракції цілісного різноманіття

сутностей, зокрема за рахунок використання засобів актуалізації та потенціалізації [18] [42]. У контексті відкрито-замкнених середовищ ТОП означає прагматико-обумовлену індивідуалізацію у ВЗСр як середовищі існування ВЗС, важливих типів відкрито-замкнених систем. Така індивідуалізація може бути як безпосередньою, так і опосередкованою. Об'єктом застосування першої є безпосередньо ЛПВ збагачуваної ВЗСр. Друга ж впливає на ЛПВ опосередковано, через прагматико-обумовлені індивідуалізації в макро і мікросередовищах ВЗСр. Результати таких індивідуалізацій можуть розглядатися біабстрактно. З одного боку, в контексті типізованого ВЗСр їх можна розглядати як існуючі у відкрито-замкненому середовищі ВЗС. З іншого, беручи до уваги згадану логіко-предметну релятивність розглядів, - кожна така ВЗС може бути в свою чергу розглянута як деяке ВЗСр. Не виключається випадок, коли ВЗСр є середовищем існування єдиної ВЗС і випадок «порожнього» ВЗСр. Загалом необхідно відзначити, що ВЗС і ВЗСр є принципово різними типами абстракції, які взаємно не зводяться одне до одного. При цьому вони доповнюють одне одного в цілісних розглядах сутностей.

Таким чином, у першому наближенні будь-яке ВЗСр може представлятися як непротирічна логічна абстракція цілісного взаємодоповнення існуючих у ній ВЗС. Концептуальну основу цього взаємодоповнення становлять згадані введення та виключення абстракції в ВЗСр. Між іншим, будь-яку ВЗС у контексті сказаного цілком природно трактувати як результат ТОП відповідного ВЗСр. Їхнє взаємодоповнення – основа біабстрактного підходу до вивчення сутностей, що розвивається у теорії дескриптивних середовищ (ДС) (див. [18] і бібліографію). Зауважимо, що відмінність типів абстракції ВЗСр і ВЗС не просто декларується, а природно реалізується тут у вигляді принципово різного використання їх у цілісних розглядах. Так сутність із точки зору ВЗСр – це інструмент, що підтримує активну роль суб'єкта – розуміння (сутності). Наприклад, сутність як ВЗСР може розглядатися як екзистон [18] [42] – *вид існування (багатьох) сутностей* (ВЗС). Та ж сутність, що трактується як ВЗС, підтримує пасивну роль результату розуміння у суб'єктній діяльності, тобто

обумовлюється суб'єктом. Зокрема, розглядається вже як актуальна сутність – представник певного ВЗСр.

У контексті сказаного необхідно особливо відзначити, що, не дивлячись на те, що найважливішими засобами здійснення будь-яких типізацій є введення та виключення абстракції, вони не можуть розглядатися як самодостатні у пізнанні сутностей як предметів вивчення. Суть використання методу введення та виключення абстракції у кожному конкретному розгляді сутності полягає у прагматико-обумовленому (отже, суб'єктивному) застосуванні цього інструменту для збагачення досліджуваного предмета. Адже саме достатньо змістовне та водночас не обтяжене специфікою збагачення предмета досліджень (а не його узагальнення чи конкретизація) – головний інструмент будь-якого розгляду.

У зв'язку з цим, хотілося б звернути увагу на принципову відмінність узагальнень і конкретизацій розглядів, з одного боку, і збагачень їх – з іншого. Хоча узагальнення і є найважливішим інструментом пізнання, видається, що догматизувати його, а тим більше – робити основною метою будь-якого дослідження – не варто. Слідування цим шляхом, вочевидь, призводить до того, що основний предмет вивчення стає всього лише «трампліном» для узагальнення та досягнення більш високого рівня загальності розгляду. При цьому основний предмет фактично залишається поза розглядами, хоча саме його збагаченню мало б бути підпорядковане проведене узагальнення. Досягнутий рівень загальності трактується як новий самодостатній предмет розглядів. Останній, внаслідок відомого закону про зворотну відповідність обсягу поняття і його змісту, вочевидь, є менш багатим у порівнянні з вихідним. Зрозуміло, що така практика, у контексті сказаного слабо відповідає процесу пізнання. Найчастіше такий підхід свідчить про недостатню прагматичну вмотивованість початкового вибору предмету вивчення. За проведення цієї стратегії не цілком абсурдним видається узагальнювати предмет будь-якого розгляду, наприклад, до поняття сутності або навіть до *дещо* як найбільш загального й такого, що включає у себе все і вся (наприклад, в розумінні [63] [64] [65]). Це максимально узагальнить

розгляд. Однак платою буде надзвичайно мала його змістовність, впоратися з якою, як зазначав Ч. Пірс (див. [64], т.2, с. 72), «... просто бракує сил». Не менш «корисним» бачиться застосування конкретизацій як виключень абстракцій предмета розгляду заради них самих. Зокрема у логічних розглядах точка зору «все є прикладна логіка» призводить до зміщення акцентів із вивчення будь-якої сутності як збагачення, а саме -конкретизації, замикання єдиної логіки досліджуваного предмета до розгляду серії прикладних логік як результатів (часто слабо мотивованих), конкретизацій цієї сутності. Це означає, що самі по собі ані узагальнення, ані конкретизації не є самодостатніми інструментами пізнання. Щоб бути такими, вони у разі необхідності повинні розглядатись у контексті прагматичної обумовленості, тобто бути підпорядкованими збагаченню уявлень про відкрито-замкненість предмету розглядів.

Зі сказаного про відкрито-замкнуту парадигму слідує принциповість ролей ВЗСР, ВЗС і ЛПВ для реальних збагачень розглядів сутностей. Тому доцільно розглянути ІТД саме у контексті відкрито-замкнутої парадигми.

Відкрито-замкнена парадигма. ІТД – суть відкрито-замкнене розуміння об’єкту ІТД.

Мета роботи, як уже зазначалося, полягає у реальному, а не номінальному розкритті природи ІТД. Змістовно кажучи, ІТД розуміється як *діяльність, спрямована на створення, дослідження і застосування ІТ-рішень*. Таким чином, ІТД асоційована принаймні з двома типами пов’язаних між собою сутностей: ІТ-рішеннями та процесами їх побудови. Природа зв’язку в контексті відкрито-замкненої парадигми розкривається наступним положенням.

Положення (про наслідковість). ІТ-рішення – наслідок відкрито-замкненого обумовлення об’єкту ІТД.

Це положення відображає природний (реальний) причинно-наслідковий зв'язок між сутностями ІТД і дозволяє аргументовано розставити акценти в її відкрито-замкненому розгляді. А саме, природним чином, відштовхуючись від реальних уявлень про ІТД, ранжувати залучені до розгляду типи сутностей,

визначивши ІТД домінантою розгляду, а ІТ-рішення(наслідок ІТД) – похідними носіями (представниками) цієї домінантності[4].

Подібні міркування щодо ВЗСр і ВЗС природно приводять нас до домінантної ролі ВЗСр і похідності ВЗС у відкрито-замкнутій парадигмі[5]. Таким чином, очевидна доцільність розгляду ІТ-рішення в контексті ВЗС, а ІТД, зокрема її логіки – у контексті ВЗСр, де ІТ-рішення реально породжуються, а не номінально надаються (декларуються).

Зважаючи на домінантне положення ІТД та значущість її для реальних відкрито-замкнених побудови, останні доцільно почати з розгляду взаємодоповнення ІТД і її результату з точки зору ЛПВ над відповідною ВЗСр і її предметним замиканням – ВЗС. Вочевидь, ВЗСр як домінанта біполя «ВЗСр, ВЗС» повинна підтримувати природні, а значить - реальні уявлення про ІТД та її результати. Як зазначалося, на самому загальному рівні розгляду розуміння ІТД знаходять своє відображення у системі основоположних загальних принципів, що відображають природу ІТД як діяльності взагалі та спеціальних, що дозволяють індивідуалізувати серед усього розмаїття діяльностей ті з них, які природно віднести до ІТ [18] [11]. Що ж до ІТ-рішень, то з огляду на відкритість цього поняття доцільним видається спиратися в розглядах, перш за все, на розкриті вище загальні інтуїтивні їх властивості. Значимість як викритих змістовних властивостей ІТ-рішень, так і системи основоположних аспектів і принципів ІТД для розкриття природи ІТД ключова. Вони дозволяють підвести під розгляд ІТД відповідний змістовний (реальний) фундамент. Його основу складає відкрито-замкнена точка зору на основний інструмент здійснення суб'єктом такої діяльності – розуміння. Тому необхідне подальше реальне збагачення як власне *розуміння* – ключового інструменту будь-якої осмисленої діяльності, так і, насамперед, *розуміння ІТД* відповідно до прагматики дослідження.

---

[4] ІТ-рішення у цій парі – номінально найбільш загальне поняття, а ІТД – найбільш фундаментальне, тобто таке поняття, яке може виступати передумовою деякого процесу, наслідком якого її результат у реальному його розумінні, тобто з урахуванням процесу його побудови.

[5] ВЗС – номінально найбільш загальне поняття, а ВЗСр – найбільш фундаментальне.

Підбиваючи підсумки, можна сказати, що основу будь-якої, зокрема і ІТД як мотивованого процесу досягнення мети, складає її розуміння. Саме розуміння має релятивну (сутесутнісну) природу, що відображає його природну здатність до еволюції. Це і є відправною точкою для формування нового уявлення про ІТД. Подальший аналіз у цьому розділі релятивної структури розуміння привів нас до формування вищенаведеної сутесутнісної парадигми, з якої випливає, що сутесутнісне відношення є частковим порядком. Далі проведений аналіз ІТД дозволив сформувати сутесутнісне положення ІТД, згідно з яким ІТД є рефлексивно-транзитивним замиканням сутесутнісного відношення. Такий розгляд реально підтримує генезис понять в їх еволюційному розвитку як розуміння як носіїв релятивності. Подальші розгляди дозволили сформувати прагматико-обумовлену конкретизацію сутесутнісного відношення - композитосутнісне відношення. А релятивність, що властива як розумінню загалом, так і ІТД має бути підтримана в їхніх розглядах у разі необхідності. Суть такої релятивності полягає у прагматико-обумовленому взаємодоповненні їхніх значимих і другорядних аспектів. Саме тому згадана відкрито-замкнута парадигма вбачається адекватною платформою для відображення згаданої релятивності.

2.2 Композитосутнісні засади АПС

Перехід від моноабстрактних до поліабстрактних розглядів, адекватно відображених у композитосутнісних відношеннях, дозволив не тільки усунути виявлені неадекватності згаданих традиційних методів, але й інтегрувати їх із методами, що розвиваються в межах денотативних підходів.

2.2.1 Денотативні аспекти АПС

Передумови широкого розвитку робіт у денотативному напрямку були закладені в роботах С. Кліні, А. Тарського, Д. Скотта, Ю.Л. Єршова, де Баккера та ін. До теперішнього часу на цьому шляху отримана величезна кількість як теоретичних, так і прикладних результатів. По суті всі вони базуються на

топологіях, індукованих тими чи іншими частковими порядками.

Функції, індуковані реальними ІТ-діяльностями (зокрема, розробками АтаПЗ), складно влаштовані. Більше того, як випливає з дескриптивного аналогу теореми Ґьоделя про неповноту, підвести під такі функції змістовну концептуально єдину денотативну базу в принципі неможливо. Тут лише можна розраховувати на створення реальної основи для індивідуалізації в універсумі таких функцій їхніх типів, які б мали прагматико-обумовлені денотативні властивості.

Безпосередньо з вищенаведеної прагматико-обумовленої аргументації визначення композитосутнісних відношень випливає, що вони можуть розглядатися в якості такої основи. Принаймні вони дозволяють індивідуалізувати типи нижче визначених ІТ (зокрема програмних) функцій, серед яких знаходять своє природне місце не тільки всі відомі нам типи традиційних ІТ та програмних функцій, але й прагматико-обумовлені нові типи принципово більш багатих нетрадиційних функцій, індукованих реальним ІТД. В якості репрезентативного прикладу таких функцій може розглядатися будь-яка функція f, що задовольняє умову:

$$f(\coprod_{n=0}^{\infty} a_n) = \coprod_{n=0}^{\infty} f(a_n),$$

де $\coprod_{n=0}^{\infty} a_n$ й $\coprod_{n=0}^{\infty} f(a_n)$ – границі послідовностей сутностей, що розуміються як найменші верхні грані в сенсі композитосутнісного часткового порядку між їхніми безпосередньо слідуючими сутностями. Домовимося їх називати *проективними функціями*. У [10] було показано, що такі функції є неперервними в топологіях, індукованих композитосутнісними відношеннями, як частковими порядками. Через теорему Тарського про структуру множини нерухомих точок безпосередньо випливає, що має місце

Теорема. Проективні функції мають нерухомі точки.

Використовуючи відомі результати, що збагачують теорему Тарського як у

напрямку її узагальнення, так і подальшої її спеціалізації можна поглибити й теорему про нерухомі точки дескриптивних функцій. Однак це вивело б нас далеко за межі розгляду предмета дисертаційної роботи. Тому обмежимося тут лише коротким зауваженням щодо обставини, відзначеної у зв'язку з наведеною теоремою.

Принципова значимість цієї теореми полягає, звичайно, не в доведенні існування нерухомих точок у проективних функцій, а в тому, що вона відкриває зовсім природній шлях для пошуку тих типів прагматико-обумовлених функцій, для яких їхнє існування є більш-менш прямим наслідком такої обумовленості. Такий підхід є не просто відмінним від відомих традиційних, а є дуальним до них, не конкуруючим, а істотно взаємодоповнюючим.

2.2.2 Композитосутнісні аспекти АПС

Представлені експлікативні збагачення ССВ забезпечують проведення ряду прагматико-обумовлених конкретизацій. Зокрема забезпечують у відповідності до прагматики цієї роботи перехід від сутесутнісної точки зору на ІТД до більш змістовної композитосутнісної парадигми ІТД.

Композитосутнісна парадигма ІТД. ІТД – суть композитосутнісне розуміння.

Ядром композитосутнісної парадигми є композитосутнісне відношення – прагматико-обумовлена конкретизація ССВ. Його основу складає узагальнене розуміння композиції, що враховує лише найбільш значущі змістовні характеристики таких операцій – адекватність, замкнутість, обчислюваність і тотальність. При цьому, як уже зазначалося, адекватність композиції обумовлює відкритість, як їхнього різноманіття, так і різноманіття відповідних операцій в цілому. Вочевидь, таке розуміння композиції, не претендуючи на всеосяжність огляду класів композицій і операцій, дозволяє коректно говорити про прагматико-обумовлений набір метазасобів маніпулювання композиціями, які

складають ядро прагматико-обумовленої, а значить, потенційно відкритої композиційної логіки (КЛ).

Відкритість композиційного й операційного різноманіть обумовлює зміну пріоритетів у композиційній і ІТ-проблематиці. А саме, робить необхідним перехід від проблеми побудови універсального відкрито-замкненого середовища ІТД до проблеми побудови адаптивних відкрито-замкнених середовищ. В основі цього переходу лежить типізація відкритої складової композиційної ВЗСр за ознакою суб'єктивної адекватності породжуваних в цьому середовищі відкрито-замкнених систем (ВЗС). Прагматико-обумовлена індивідуалізація наборів базисних композицій із наступним замиканням композиційної логіки ВЗСр в індивідуалізованих суб'єктом базисних композиціях є тим засобом, який обумовлює породження обумовлених прагматикою типів операцій. Тому природно назвати цей засіб *типізатором*, а предметну діяльність, яку він презентує – (прагматико-обумовленим) типізуванням. Таким чином,

$$\begin{cases} \textit{типізатор} - \textit{сутність, що базисно обумовлює операцію;} \\ \textit{тип} - \textit{сутність, що обумовлюється типізатором.} \end{cases}$$

Введений цією експлікацією механізм прагматико-обумовленої типізації складає основу концепції адаптивного процесонального середовища (АПС). Це середовище підтримує маніпулювання прагматико-обумовленими типами, являючи собою, таким чином, середовище адаптивних типізацій (САТ) – несуперечливу логічну абстракцію цілісного різноманіття прагматико-обумовлених типів.

Змістовно кажучи, значимість експлікації як типізування, так і власне типу полягає в тому, що кожен представник роду типів фактично є інструментом прагматико-обумовленого опису в широкому сенсі[6] генетичної (композиційної) природи найрізноманітніших сутностей. Тому таких представників роду типів

[6] Опис тут не звужується до рівня мовного, вербального та взагалі будь-якого окремого виду описів. Ближчий за змістом термін «бачення», «усвідомлення» і навіть «розуміння». В цілому, опис тут розуміється в сенсі об'єднання найрізноманітніших таких трактувань.

природно називати *дескрипціями*, те, що їх обумовлює – *дескриптами*, а відповідну предметну діяльність – *дескриптуванням*.

$$\begin{cases} \textit{дескрипт} - \textit{сутність, що дескриптивно обумовлює тип}; \\ \textit{дескрипція} - \textit{суть, що обумовлюється дескриптом}. \end{cases}$$

Таким чином, вирішується відоме протиставлення об'єктивної відкритості засобів композиціонування як генетичних структур початковій обмеженості будь-якого, навіть широко зафіксованого класу представників роду композицій. Це протиріччя обумовлене самою природою генезису як суб'єктообумовленого процесу, який навіть за бажанням не можна «вписати» в будь-які, нехай найширші, але жорсткі межі. Таким чином, КСО «композит, композиція» являє собою принципово нову композиційну логіку. Логіку, невід'ємним атрибутом якої є відкритість її носія, об'єктивно зумовлена природною суб'єктністю його представників. Цей носій цілком природно розглядати як універсум композицій (УК) – несуперечливу логічну абстракцію цілісного різноманіття композицій. Це дозволяє індивідуалізувати клас загальнозначущих, а значить, логічних в УК композицій, які в силу їхньої загальнозначимості природно назвати *метакомпозіціями*.

Відкрито-замкнута парадигма в композиційній проблематиці зумовлює доцільність розкриття композиційних метазасобів як логічного фундаменту розгляду. В основу такого розкриття покладемо відому парадигму «розділяй та володарюй» (лат. divide et impera), яка пройшла серйозну перевірку часом і є системоутворюючою основою природничо-наукових уявлень про вирішення задач. Змістовно кажучи, ця формула презентує агрегацію виду «ціле-частина» як засіб прагматико-обумовленого дослідження сутностей. Аналіз практики його застосування дозволяє індивідуалізувати засоби виду підстановок (суперпозицій) і застосувань (аплікацій) в якості основних інструментів агрегування. Тому саме серед суперпозицій та аплікацій доцільно шукати сьогодні загальнозначущі засоби (метазасоби) композиційної логіки. Таким

чином, розглянута *композиційна логіка ІТД* (КЛІТД) представляється у вищеозначеному носієм сенсі – універсумом композицій, що розуміється як несуперечлива логічна абстракція цілісного різноманіття композицій із введеними на ньому метакомпозіціями типу суперпозицій та аплікацій.

Проведені побудови суттєво збагачують розуміння ІТД. У першу чергу це стосується конкретизації ССВ як домінанти відкрито-замкнених розглядів ІТД до КСВ – основи логіко-предметного збагачення розуміння ІТД. Саме це забезпечило перехід від сутесутнісної до композитосутнісної парадигми ІТД та створило реальну основу для розбудови концепції адаптивних процесональних середовищ (АПС).

2.2.3 Композитосутнісне відношення – ядро АПС

Тут на базі отриманих раніше результатів здійснюється розгортання поняття адаптивного процесонального середовища як відкрито-замкненого середовища, що підтримує ІТД як у сенсі процесу, так і в сенсі її результату. У своїй основі ці дослідження зводяться до подальшого прагматико-обумовленого збагачення композитосутнісної парадигми.

У першому наближенні адаптивне процесональне середовище (АПС) є інтеграційним середовищем, що поєднує, а не протиставляє різноманіття прагматико-обумовлених точок зору на об’єкти ІТД, тобто є відкрито-замкнутим. Концептуально єдину основу розглядів складає поняття композитосутнісного відношення.

Підставою для вибору будь-якої точки зору на сутності ІТ-сутнісної платформи (ІТСП) як концептуально єдиного носія АПС є розгляд їх у двох взаємодоповнюючих сенсах. Один із них полягає в тому, що сутності розглядаються в пасивній ролі – розглядуваної сутності, тоді як інший сенс пов'язаний із її активною роллю – засобом моделювання сутностей.

Така взаємодоповнюваність обумовлює концептуально єдиний погляд на ПСП як на систему, в якій явно індивідуалізовані сукупності розроблюваних сутностей і засобів їхнього моделювання. Спочатку на засоби моделювання, як і

на різноманіття сутностей, ніяких обмежень не накладається. Тобто маємо справу з універсумом засобів побудови (УЗП), що включає будь-які прийоми, способи та методи розробки сутностей. Говорячи більш строго, УЗП розуміється як несуперечлива логічна абстракція цілісного різноманіття різних засобів розробки сутностей. Домовимося наочно представляти цю систему у вигляді:

ІТСП = <НПС, УЗР>

де НПС – це носій сутностей, можливо збагачений прагматико-обумовленими точками зору на них.

У межах ІТСП, що еволюційно розвивається в роботі, важливі не стільки просто сутності, скільки засоби їхньої розробки. Пов'язано це, перш за все, з тим, що поряд із активною роллю в ІТСП вони можуть виступати в пасивній ролі, тобто поряд із засобом ІТД бути також предметом ІТД.

Можливість таких сутностей виступати у двох ролях (сенсах) дозволяє цілком природно говорити про невироджені (на відміну від традиційних) застосування їх самих до себе. Це не тільки істотно збагачує (при цьому, особливо важливо, що несуперечливо) добре відомий принцип самозастосовності, який лежить у основі λ-числення [66], але і робить його реальним прагматико-мотивованим засобом інтенсіалізації еволюційного розвитку ІТСП.

Для того, щоб цю можливість втілити реально, слід, насамперед, конкретизувати НПС як універсум об'єктів, що ототожнюється далі з універсумом рішень (УР), в якому явно індивідуалізований універсум предметів (УП) як сутностей із УЗП, що виступають у пасивній ролі. Для наочного узагальнення цю конкретизацію представимо у вигляді:

НПС = <УР, УП>.

Тоді, переслідуючи ту ж мету, цілком природно ІТСП конкретизувати у вигляді:

ІТСП = << УР, УП>, УЗП>

де УЗП як сукупність сутностей екстенсіонально дорівнює сукупності предметів, але інтенсіально – роль їхня вже активна, а не пасивна, тобто принципово інша.

Індивідуалізація носія ІТСП загалом не виключає того, що УП у ньому відсутній. Зазвичай так і відбувається. Ролі (сенси) сутностей не залучаються до розгляду, а терміни «об'єкт» і «предмет» трактуються як синоніми. До певного часу це було доцільно робити. Справа в тому, що аспекти сутностей, що залучалися до розгляду, були настільки прості, що втягувати у розгляди ще і їхні ролі, а значить, вносити об'єкт і предмет у структуру носія, не було необхідності.

Становище докорінно змінилося з постійно зростаючим ускладненням програмно-апаратних розробок. Адже таке ускладнення за необхідністю висунуло на перший план необхідність врахування як взаємодоповнюваності принципово різних аспектів сутностей у межах їхнього носія, так і взаємодоповнюваності останнього із сутностями УЗП, погодженими з цими аспектами в межах ІТСП. Адже без урахування такої двоєдиної взаємодоповнюваності говорити про збереження інвестицій у сучасних програмно-апаратних розробках не доводиться.

Більше того, з усього сказаного вище випливає, що і врахування двоєдиної взаємодоповнюваності недостатньо. Це лише необхідна умова. Справа в тому, що з вищевикладеного витікає, що ІТСП як << УР, УП>, УЗП> щонайменше з метою збереження інвестицій має виступати як основа універсального середовища інтеграції (УСІ). При цьому УСІ повинно знаходитися у відношенні взаємодоповнюваності із засобами еволюційного розвитку (ЗЕР) ІТСП. Адже наочне уявлення виду:

$$УСІ = <ОС, ЗЕР>$$

де ОС = ІТСП, як і ІТСП = <НС, УЗП>, всього лише форма. Що ж до головного - змісту, то його основу складає взаємодоповнюваність як характерний атрибут УСІ. Наочно триєдина взаємодоповнюваність представлена у вигляді:

$$УСІ = <<< УР, УП>, УЗП>, ЗЕР>.$$

Резюмуючи викладене, є всі підстави зробити висновок, що прагматико-мотивовані релятивізації будь-яких розроблюваних сутностей цілком можуть розглядатися як абсолютно природні експлікати розроблюваних сутностей у ролі експліканд.

Зрозуміло, що все сказане не більше ніж загальний погляд на внутрішні та зовнішні властивості АПС як триєдиної системи взаємодоповнення основою якої є УСІ. Проведені дослідження досить переконливо мотивують, що ні про яке серйозне збереження інвестицій у сучасних інформаційно-технологічних системах, зокрема ІТС, не може бути й мови без явного врахування цих властивостей у межах концептуально єдиної інтеграційної релятивізації. Щонайменше внутрішні властивості в цій релятивізації мають бути враховані на рівні двоєдиної взаємодоповнюваності, втіленої в структурі носія ІТСП, а зовнішні – на рівні індивідуалізації ІТСП в якості основи триєдиної взаємодоповнюваності в межах УСІ.

Дотепер як УЗП, так і ЗЕР навмисне інкапсульовувалися. Настала пора явним чином втягнути їх у розгляди у контексті композитосутнісної направленості даної роботи. З отриманих вище результатів дослідження безпосередньо випливає висновок про те, що як сутності з УЗП, так і з ЗЕР за необхідністю є носіями прагматико-обумовленої точки зору на генезу сутностей. Тобто є композиціями у межах відповідної прагматики. Основне ж призначення УСІ – наповнення реальним змістом, надання обґрунтованого проведеними дослідженнями адекватного розуміння реалізації засадничого принципу цієї роботи – принципу процесональності. Основними складовими УСІ є прагматико-обумовлені відкрито-замкнуті середовища (ВЗСР) як засіб реалізації базових композитосутнісних відношень, індукованих УЗП та ЗЕР. Саме ж УСІ являє собою ВЗСР, що підтримує рефлексивно-транзитивне замикання композитосутнісного відношення як результату взаємодоповнення, зокрема об’єднання, базових відношень.

З результатів, отриманих у попередньому підрозділі випливає, що як УЗП, так і ЗЕР є відкритими системами. Це унеможливлює здавалось би природний

висновок про те, що АПС є УСІ. Адже УСІ у силу відкритості його основних складових є занадто малозмістовним. А значить потребує прагматико-обумовленого збагачення. Змістовна його суть полягає у тому, що хоча УЗП та ЗЕР і не можуть, у силу своєї відкритості, бути несуперечливо представлені вцілому, ніщо не заважає розглядати окремі прагматико-орієнтовані класи генетичних структур, отримані як результати застосування окремих прагматико-обумовлених засобів еволюційного розвитку до окремих, частіше за все скінчених наборів прагматико-обумовлених базових композицій. Конкретніше говорячи, хоча теорема про нерухомі точки, наведена у пункті 2.2.1 і дає уявлення про ЗЕР та УЗП як про сукупності рішень рівнянь типу $a = f(a)$, де a – деяка сутність (наприклад, для ЗЕР – композиція, для УЗП – абстрактна, іменна або метаіменна функція), а f – ІТ-функція, однак сама сукупність таких рішень є відкритою системою. Тому АПС цілком обґрунтовано бачиться сьогодні як конкретизація УСІ у частинах ЗЕР та УЗП шляхом обмеження їх прагматико-обумовленими початковими умовами.

Що стосується засобів еволюційного розвитку, то сучасне їх розуміння, індуковане відомою парадигмою вирішення задач «розділяй та володарюй» пройшло серйозну перевірку часом. З огляду ж на результати попереднього розділу про класифікацію різноманіття композицій можна зробити обґрунтований висновок про те, що ЗЕР складають композиції аплікації та суперпозиції. Аплікація для будь-якої композиції K і функції f ставить у відповідність нову функцію, яку позначатимемо $K(f)$, що являє собою результат застосування композиції K до аргументу f. Композиція S^n є суперпозицією n-арних функцій [61] [67] [7]. Змістовно кажучи, перша підтримує покрокове зведення складно влаштованих композицій до більш простих (аспект володарювання), друга ж є засобом специфікації прагматико-обумовлених поділів сутностей на взаємодоповнюючі частини. Початкові ж обмеження для УЗП індуковані прагматиками вирішуваних ІТ-задач.

Таким чином, із вище наведеного випливає обґрунтований висновок про те, що АПС являє собою УСІ, обмежене зазначеними початковими умовами. Будемо називати таке обмеження ІТ-релятивізацією УСІ. Зафіксуємо сказане у вигляді головної тези.

Головна теза. АПС експлікативно зводиться до ІТ-релятивізації УСІ.

Всі подальші розгляди будуть підпорядковані подальшому збагаченню цієї тези, як на рівні прагматико-семантичних досліджень, так і програмних побудов.

Отримавши ці результати, можна ще раз більш конкретно розглянути АПС як ВЗСР та узагальнити наведені вище розгляди.

2.2.3.1 Замкнена частина АПС

Замкнена частина адаптивного процесонального середовища являє собою найбільш значимі загальні уявлення про організацію ІТП. Тому чи не єдиним джерелом конкретики тут виступає прагматика – найбільш значущий, загальний та незалежний від будь-якої специфіки аспект вирішення задач. А принцип обумовленості диктує основну ціль прагматичних досліджень – віднаходження природи генезису відповідного рішення задачі. Мова йде про віднаходження генетичних структур як *реальних* уточнень представлень розробника про способи інтеграції рішень підзадач. За принципом композиційності генетичні структури програм – композиції. Таким чином, результатом прагматичних досліджень є сформоване розуміння адекватних способів генезису відносно більш складних рішень із відносно простіших. Представлення цього розуміння відповідно до принципів функціональності та композиційності у вигляді класу алгебраїчних операцій і прагматико-обумовлене виділення в ньому базової сукупності композицій є ключовою складовою реалізації принципу підпорядкованості у відношенні прагматико-семантичних досліджень. Таким чином основу закритої частини будь-якого адаптивного процесонального середовища складає прагматико-обумовлена конкретизація загального поняття композиції у вигляді ряду адекватних до умов застосування генетичних структур різного рівня загальності.

2.2.3.2 Відкрита частина АПС

ІТД пов’язана з розглядом сутностей, які відображають основні аспекти, властивості, відношення, якими характеризується предметна область. Відкритою частиною адаптивного процесонального середовища є прагматико-обумовлене різноманіття сутностей, які є можливими продовженнями закритої частини, індукованими відповідними логіко-предметними відношеннями. Це «різноманіття» не є якимось конкретним різноманіттям, а являє собою несуперечливу логічну абстракцію цілісного різноманіття сутностей, яка є його суттю в даному адаптивному середовищі. Це дозволяє не вносячи до розгляду протиріччя усього різноманіття, працювати з його несуперечливою суттю (прагматико-обумовленою абстракцією), коректна (несуперечливо) продовжуючи його до конкретних представників або класів різноманіття в потрібний час в потрібному місці розгляду. Іншими словами представники відкритої частини відкрито-замкненого середовища або їх класи трактуються тут як можливі результати (наслідки) відповідних процесів предметного, значить несуперечливого, продовження (замикання) логічного ядра відкрито-замкненого середовища (його замкненої частини). Суттю же самих процесів продовження є третя основоположна частина відкрито-замкнутих середовищ – його логіко-предметне (композитосутнісне) відношення.

2.2.3.3 Композитосутнісне (Логікопредметне) відношення.

Дана складова відкрито-замкнених середовищ грає справді головну роль. Саме ЛПВ з двох різних сутностей (відкритої та замкненої частин) створює принципово нову сутність – відкрито-замкнене середовище, в якій вони і виконують свої ролі відкритої та замкненої частин. Змістовно кажучи, ЛПВ представляє причинно-наслідковий зв’язок між властивостями та сутностями, які їх мають. В цьому сенсі ЛПВ є відношенням, в якому сутність має деяку властивість. У випадку з КСВ властивістю є композиційність. ЛПВ є несуперечливою логічною абстракцією цілісного різноманіття згаданих

«володінь властивостями», як процесів продовження композицій до сутностей, суттю яких вона є.

У цьому підрозділі на базі отриманих в попередніх розділах результатів здійснено розгортання поняття адаптивного процесонального середовища, як відкрито-замкненого середовища, що підтримує ІТД як у сенсі процесу, так і в сенсі її результату. У своїй основі ці дослідження зводяться до подальшого прагматико-обумовленого збагачення композитосутнісної платформи. У першому наближенні, АПС є інтеграційним середовищем, що поєднує, а не протиставляє різноманіття прагматико-обумовлених точок зору на об’єкти ІТД. Концептуально єдину основу розглядів складає поняття композитосутнісного відношення. Таку ж основу складає сутесутнісне відношення для універсального середовища інтеграції. З проведених раніше в розділі побудов випливає обґрунтований висновок про те, що АПС являє собою УСІ, обмежене зазначеними початковими умовами. Таке обмеження названо релятивізацією УСІ, а сказане зафіксовано у вигляді головної тези: АПС експлікативно зводиться до ІТ-релятивізації УСІ. Всі подальші розгляди будуть підпорядковані подальшому збагаченню даної тези, як на рівні прагматико-семантичних досліджень, так і програмних побудов.

2.2.4 Інтуїтивні властивості ІТ-рішень

Залежно від тих чи інших прагматичних вимог на передній план висуваються найрізноманітніші риси, характерні для більш-менш широких класів ІТ-рішень. Однак, які б не були ці вимоги, поряд із зазначеною вище властивістю визначеності з-поміж інших зазвичай зустрічаються властивості, зазначені нижче.

1. Фінітність. Будь-яке ІТ-рішення як достатньо деталізоване представлення рішення задачі – скінчений (фінітний) об'єкт як за формою, так і за змістом: отримання результату його виконання пов'язане лише зі " скінченими даними" і "скінченим часом". Так трактована властивість фінітності є

загальновизнаною необхідною умовою будь-якої механічної (такої, що не потребує додаткових роз’яснень) реалізованості.

2. Масовість. Ця властивість розуміється тут як можливість застосування ІТ-рішення до множини різних вихідних умов, даних. При цьому не вимагається, щоб останні були обов’язково потенційно нескінченими. Таким чином, властивість масовості ІТ-рішення за необхідністю тлумачиться тут дещо ширше, ніж це прийнято стосовно програм у багатьох роботах по спеціальним алгоритмічним системам (частково-рекурсивним функціям, машинам Тюрінга, нормальним алгоритмам Маркова і т.д.). Пов'язано це з тим, що у розгляд залучаються ІТ-рішення як із потенційно нескінченними множинами вихідних даних, так і зі скінченними.

3. Результативність. Із кожним ІТ-рішенням пов'язується множина можливих, як проміжних, так і остаточних результатів його реалізації (виконання, впровадження). Як і вище, ці множини не обов'язково потенційно нескінченні. Результативність ІТ-рішення забезпечується тим, що будь-який із них має явно або неявно задані засоби, що дозволяють серед проміжних даних, одержуваних на тих чи інших етапах його виконання, відбирати остаточні (підсумкові, результативні).

4. Дискретність. Розуміється у тому сенсі, що процес виконання ІТ-рішення на будь-яких вихідних даних здійснюється покроково у дискретному часі.

5. Редукційність. Це властивість трактується у тому сенсі, що загалом будь-який «складний» (головний) ІТ-рішення являє собою інтеграцію відносно «простих» його складових ІТ-рішень, обумовлених кроками реалізації основного ІТ-рішення.

6. Детермінованість. Полягає у тому, що дані, крім вихідних, отримані у певні моменти часу в процесі реалізації ІТ-рішення, однозначно визначаються даними, отриманими у попередні моменти часу.

7. Визначеність. Виконання ІТ-рішення є механічною дією. Це означає, що виконавець не потребує для реалізації ІТ-рішення додаткових роз’яснень.

Окремим важливим випадком визначеності є властивість обчислюваності функції у сенсі можливості її автоматної реалізації.

8. Структурність. Означає, що ІТ-рішення є структурною сутністю. Тобто коректно говорити про його структуру.

9. Генетичність. Щодо будь-якого ІТ-рішення коректно говорити про його генезис. Ця властивість є простим наслідком властивостей структурності та редукційності.

10. Реалізованість. Щодо будь-якого ІТ-рішення коректно говорити про його реалізацію. Ця властивість взаємодоповнює згадані вище властивості фінітності, масовості, змістовності, редукційності і т.д.

Наведений перелік не є виключним, а всі перераховані властивості носять гранично загальний, а відтак - малозмістовні характер. Щоб підвищити рівень конкретності, слід розглядати ІТ-рішення вже не як свого роду «чорні ящики», а як сутності, що мають певну структуру не тільки «в принципі», а при залученні до розглядів як конкретних типів структур, так і, що найважливіше, процесів, способів, методів і засобів набуття ІТ-рішеннями структурності того чи іншого виду. Але для цього необхідно залучати до розгляду вже не стільки ІТ-рішення або ІТД самі по собі, скільки їхнє взаємодоповнення. Саме воно складає основу адаптивних процесональних середовищ (АПС). Тому залучення такого взаємодоповнення має носити реальний, тобто прагматико-обумовлений, характер.

2.3 Адаптивні середовища для вирішення ІТ-задач

Через раніше згадані кризові прояви в індустрії інформаційних технологій, надзвичайно актуальним стало розробити з кожним разом дедалі більш нові підходи до вирішення ІТ-задач, які б підтримували сам ІТП. При більш загальній постановці ІТ-задачі, справедливим буде твердження, що дійсність як носій будь-якої діяльності, зокрема ІТД, нерозривно пов'язана з тими чи іншими сутностями. Через те, що дійсність неможливо зрозуміти, не пізнавши як самі сутності, якими ми оперуємо, так і природу їхнього зв'язку, неможливо також

розв'язати ІТ-задачу, не пізнавши самі сутності та природу їхнього зв'язку. У першому наближенні ця природа обумовлена розділенням сутностей, що використовуються в ІТП, в контексті їхньої важливості для конкретної проблеми на головні та другорядні. Цей розподіл формує фундамент концепції відкрито-замкнутих систем (ВЗС) [36] [42] [68], що буде використовуватись для побудови адаптивних середовищ.

Специфіка переважної більшості сучасних задач така, що виконати розподіл на головне та другорядне раз і назавжди неможливо. Найчастіше таке рішення складається з багатьох кроків. Крім того, самі ці кроки можуть бути складно пов'язані між собою, і на сьогодні питання про їхній закінчений огляд навіть не стоїть. Тому єдине, про що зараз можна говорити, так це про інструмент, який хоча б підтримував кроки ІТД в "правильному" напрямку. Останнє означає, що кожен наступний крок має як принаймні не руйнувати досягнутий рівень розділення на головне та другорядне і бути коректним щодо такого розділення. Такий інструмент був би універсальним засобом для збагачення розгляданих сутностей у процесі отримання рішення. Тому його природа складна та пов'язана з сутностями вирішуваних проблем та задач. Добре відомо, що існує нерозривний зв'язок між кроками ІТП та його результатом [19] [20] [67] [36] [42]. Головну роль у виявленні суті такого зв'язку грає роз'яснення логіко-предметного взаємозв'язку відкрито-замкнутих середовищ (ВЗСр) і ВЗС як основного елементу покрокового збагачення сутностей, що розглядаються в ІТП.

У першому наближенні суть такого підходу до побудови апаратного та програмного забезпечень складається у досягненні прагматико-обумовленого компромісу між замкнутою та відкритою точками зору на них. Відомо, що поряд із беззаперечними перевагами як замкнута, так і відкрита точки зору мають принципові недоліки, які все більш домінують. Це робить актуальним формування такої точки зору, котра була б вільна від таких недоліків, зберігаючи при цьому переваги як закритої, так і відкритої точок зору. Цю точку зору, яка зберігає переваги, назвемо відкрито-замкнутою. Перш за все, для підтримання компромісу між такими точками зору, необхідно знайти причини недоліків [68].

Таким чином, відкрито-замкнута система має не тільки розділяти першочергове та другорядне, а ще й встановлювати між ними взаємозв'язок. Розділ такий є не номінальним, а реалізованим у самому понятті ВЗС шляхом різного застосування головних і другорядних сутностей. Головні сутності формують замкнуте ядро ВЗС, яке обумовлене прагматикою конкретної задачі, а другорядне формує множину можливих замикань ВЗС і є її відкритою частиною. Це дозволяє розглядати на ВЗС як на прагматико-залежний біабстрактний зв'язок відносно автономізованих замкнутої (логічної, загальної) та предметної частин системи як принципово різних типів абстракції. Логіко-предметна направленість є визначальною для концепції відкрито-замкнутих систем. При цьому як логіка, так і предмет тут розглядаються відносно та можуть бути самі розглянуті з позиції логіко-предметних відношень. Саме така відносність дозволяє говорити ще й про відкрито-замкнуті середовища (ВЗСр) як середовища існування відкрито-замкнутих систем.

Саме з позиції відкрито-замкнутих середовищ у даній роботі розглядаються адаптивні процесональні середовища. Адаптивність у цьому випадку досягається через ітеративне поповнення ВЗСр через породження нових композицій та застосування композицій для створення нових функцій. Цей процес можна схематично зобразити (рис. 2.1). Таке поповнення відбувається доти, доки з’явиться така відкрито-замкнута система (ВЗС), котра буде зручною для вирішення необхідного класу задач.

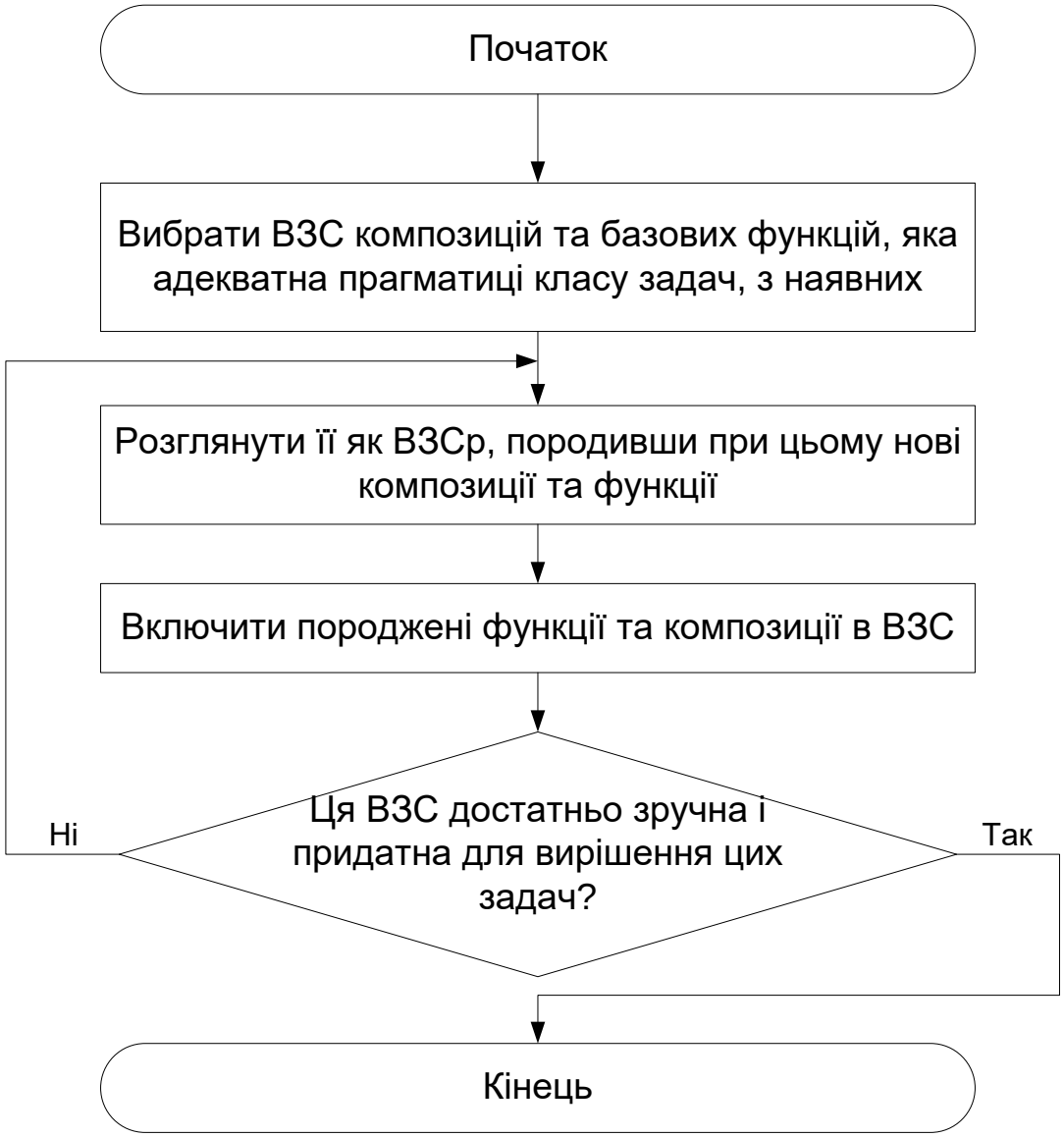


Рис. 2.1 Схема адаптації адаптивного процесонального середовища.

Адаптивний підхід розглядає три наступні типи операцій:

- функції;
- композиції;
- метакомпозиції.

Всі вони мають ієрархічну структуру (рис. 2.2).

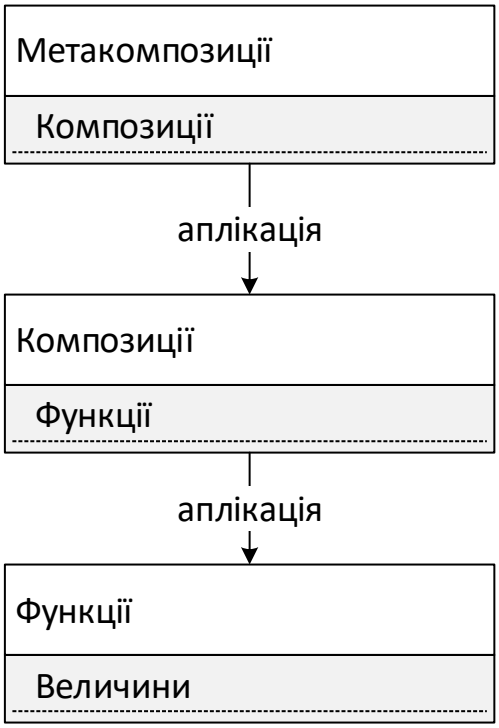


Рис. 2.2 Структура і відношення елементів адаптивного процесонального середовища.

Ці структури пов’язані між собою операцією аплікації, яка дозволяє перейти на нижчий рівень ієрархії:

- аплікація метакомпозиції продукує композицію;
- аплікація композиції продукує функцію;
- аплікація функції продукує елемент носія (величину).

У роботі наведена демонстрація роботи такого підходу на прикладі з використанням мови Verilog. Результуюча функція, композиції та метакомпозиції будуються в мові Verilog. Рішення оформлюється в зручну для використання форму та далі може легко інтегруватись із іншими обчислювальними системами за необхідністю.

Слід зазначити, що у всіх функцій і композицій має бути однакова арність. Ця вимога ставиться для того, щоб можна було їх композиціювати між собою. Не обов’язково арність функцій має бути рівною арності композицій, але в межах свого класу вона має бути однаковою. Тоді як композицій та функцій може бути безліч, обґрунтована необхідність тільки однієї метакомпозиції, а саме

суперпозиції. Вже за цих умов адаптивний підхід задовольняє визначення програмування як «породження та застосування композицій».

2.3.1 Підходи до розробки реконфігурованих систем

Сучасні підходи до розробки реконфігурованих систем відносяться до, найвищого, системного рівня [69] ієрархії процесу розробки АЗ цифрових логічних пристроїв, де виділяють щонайменше три рівні [70]: фізичний, рівень регістрових передач і системний рівень, на якому розробник абстрагується як від фізичних особливостей майбутньої схеми, так і від примітивних логічних пристроїв на зразок регістрів та логічних вентилів.

Для попередньої оцінки вимог до продуктивності обчислювальної платформи виконується аналіз простору проектних параметрів (design space, простір, вимірами якого є важливі характеристики майбутнього рішення; розробник має на меті знайти глобальний мінімум функції можливих реалізацій АЗ за умови, коли менше значення характеристики краще) перед вибором архітектури [71]. І справді, розробка на високому рівні абстракції дозволяє збільшити швидкість розробки завдяки тому, що не потрібно проводити затратну низькорівневу симуляцію.

Методика Y-chart [72] є засобом організації процесу розробки АЗ та полягає у виділенні трьох різних аспектів розробки цифрової системи: поведінкового, структурного та геометричного. Всі вони рівноправні між собою та представляють різні погляди на одне і те ж рішення. Кожен з аспектів може розглядатись на різних рівнях абстракції, наприклад: рівні регістрових передач, алгоритмічному рівні (опис та специфікація взаємодії різних підсистемам) і архітектурному рівні (вимоги до системи, основні концепції, які мають забезпечити задоволення цих вимог). Є варіанти з більшою кількістю рівнів абстракції. Поведінковий аспект описує функціональну поведінку системи, структурний аспект відображає зв’язки систем і під систем рішення на різних рівнях абстракції, геометричний аспект відповідає на питання про розмір, розміщення та форму підсистем на кристалі. Давши відповідь на те, що

знаходиться на кожному рівні абстракції кожного аспекту, розробник матиме Методика Y-chart дозволяє визначити чітку межу між програмою поведінки й архітектурою (виділеними програмованими та реконфігурованими ресурсами) завдяки явно вираженому етапу відображення системи на конкретну архітектуру, що дозволяє узгодити програмну (обчислення та комунікаційні події) й архітектурну (часові характеристики та наслідки від подій у програмі) моделі [73] [74].

Ще одним підходом до конструювання реконфігурованих систем є конструювання на базі платформ (platform based design, PBD), особливістю якого є повторне використання IP (intellectual property) ядер [75]. Суть його полягає в тому, що в основу рішення закладена деяка обчислювальна платформа, наприклад, Texas Instruments OMAP, Philips Viper Nexperia, і її функціонал розширюється IP ядрами від виробника платформи чи від третіх осіб. Це дозволяє значно скоротити витрати на виробництво електронних продуктів в рази та автоматизувати в деякою мірою процес розробки АЗ. Крім того, такі платформи дозволяють організувати стек із платформами більш високого рівня, які, наприклад, призначені для розробки програмного забезпечення. У цьому випадку верхня платформа стеку є джерелом умов та обмежень для нижньої платформи, а нижня платформа в цій взаємодії визначає продуктивність та швидкодію стеку. Хоча, як зазначено в [76], PBD – це компромісне рішення при розробці систем на кристалі, необхідно мати на увазі, що зміни архітектури зумовлюють значну кількість роботи над верифікацією та налаштуванням. Наприклад, зміна розрядності процесора призводить до розробки нового інтерфейсу з пам’яттю, з подальшою його стандартизацією та верифікацією. Крім того, через проблеми з сумісністю та продуктивністю найновіші розробки не можуть мігрувати повністю на реконфігуроване апаратне забезпечення [77]. PBD-підхід до розробки з побудовою проблемно орієнтованих процесорів передбачає розробку унікального набору команд, інтерфейсу процесора і т.д. на основі базової архітектури [78]. Справді, в деякою мірою процесор із проблемно орієнтованим набором команд (application-specific instruction-set processor, ASIP)

потенційно може досягнути необхідного компромісу між гнучкістю та продуктивністю (гнучкість базового процесора та продуктивність, яка досягається виконанням проблемно-орієнотваного набору команд). Наприклад, високопродуктивний процесор Tensilica Xtensa, оптимізований для обробки сигналів, може включати широкий діапазон попередньо верифікованих DSP модулів від прискорювача операцій із плаваючою комою до модуля на 16 паралельних операцій MAC. Крім того, SoC-платформи в PBD дозволяють поєднати гнучкість програмного забезпечення (стандартні програмовані ядра, DSP і ASIC) із продуктивністю реконфігурованої логіки. Таким чином, вони краще адаптовані до різних застосувань та здатні краще забезпечувати сумісність [79].

Проблемно орієнтовані платформи стають дедалі більш розповсюдженими. Більшість DSP сьогодні використовуються для обробки даних під час радіопередачі, зокрема в мобільних телефонах, а нові застосування збільшать вимоги до продуктивності та споживання електроенергії. Це призведе до збільшення використання не платформо-орієнтованих вбудованих систем, а до систем, що орієнтовані на певне коло задач (domain oriented) [79]. Те, що в окремих ситуаціях, наприклад, із так званими бутилковими горловинами (вузькими місцями типу прийом-передача радіосигналу, швидка обробка сигналу по протоколу і т.д.), дійсно потрібні проблемно-орієнтовані системи – це факт. Точки їхнього використання обумовлені архітектурним рішенням системи. Якщо в точці потрібно виконувати лише додавання одиниці до даного, що прийшло, і більше ніяких інших, то невиправданим із погляду архітектури є застосування в ній процесору, який підтримує всю арифметику. Хоча економічно це може бути виправдано. Наприклад, є серійна дешева мікросхема арифметики, але немає мікросхеми додавання 1. Але проблемно-орієнтовані архітектури не можуть замінити собою універсальні архітектури. Потрібен симбіоз. ПОА є не більш ніж засобом оптимізації універсальних рішень. І справді, останні комерційно успішні конфігуровані платформи є не орієнтовані на задачу (application specific), а більше орієнтуються на певне коло застосувань. На сьогодні нам в

якості проблемно-орієнтованих платформ нам пропонують як FPGA, так і реконфігуровані процесори та системи на кристалі [80] [81]. FPGA, які призначені для вузькоспеціалізованих (embedded) рішень, із вбудованою FLASH пам’яттю вдало балансують між продуктивністю та енергоспоживанням [82]. До того ж, у великих партіях вони можуть успішно застосовуватись у продуктах, де ціна має велике значення. Швидкий час розробки продуктів на їхній базі та гнучкість можуть стати вирішальною причиною вибору саме FPGA, а не фіксованої апаратної платформи. Наприклад, FPGA Actel ProASIC3, Fusion та Igloo, що використовують flash пам'ять також можуть містити процесорне IP ядро. В CoreMP7 від Alcatel є 32-бітне ARM ядро, операційна система реального часу та багато різних периферійних пристроїв, що можуть бути підключені по шині AMBA, що забезпечує їхню взаємодію.

Моделювання на рівні транзакцій (transaction level modelling, TLM) є ще одним підходом до розробки складних цифрових систем. Він передбачає налагодження механізму комунікацій та інтерфейси на високому рівні абстракції шляхом виключення з процесу моделювання самих реалізацій підсистем. Натомість моделюється тільки їхня поведінка та процес комунікації (транзакції) між ними. Це обумовлено тим, що всю структуру міжз'єднань і компонентів реконфігурованих систем все важче включати, а процес моделювання ускладняється, що призводить до зростання вимог до продуктивності засобів моделювання. Виклик функцій у TLM моделює комунікацію й одночасно приховує варіанти її реалізації (топології міжз'єднань, комунікаційні механізми та примітиви синхронізації) [83]. Крім того, що TLM допомагає в симуляції, вони також спрощують оптимізацію та моделювання топологій і протоколів [82] [83]. Також необхідно враховувати реактивну природу вбудованих систем [84]. Симуляція реактивної транзакції (транзакція може бути припинена або переініціалізована ще до її закінчення) може формально довести коректність обраного способу реалізації (в конструкції немає петель і вона може завжди відреагувати на звернення).

Реконфігуровані платформи на базі FPGA швидко стали основним рішенням для забезпечення гнучкості програмного забезпечення та продуктивності апаратного забезпечення при розробці систем. Системи на FPGA більш швидкодіючі, аніж системи на базі процесора та програмного забезпечення. Вони одночасно і менш статичні порівняно з рішеннями на ASIC зі швидшим часом розробки. На жаль, висока ціна також є бар'єром для використання FPGA у вбудованих системах. Ціна пристроїв на FPGA у масовому виробництві все ще вище ціни ASIC. Хоча різниця зменшується з поступовим завоюванням ринку FPGA. Нові покращення в архітектурі FPGA дають привід сподіватись, що нове покоління FPGA може значно зменшити споживання електроенергії та площі кристалу [85]. Так нові flash FPGA споживають менше електроенергії порівняно зі SRAM [86]. Нові архітектури, що оперують більш високими рівнями абстракції, включаючи маршрутизації на шинах, логічні блоки (АЛУ та DSP) також обіцяють покращити продуктивність і ефективність [87]. Іншою важливою особливістю нових FPGA архітектур є можливість інтегрувати один або більше процесорів загального призначення [88] [89].

Сучасні FPGA мають достатньо ресурсів для того, щоб повністю реалізувати систему на кристалі. Але сучасні задачі просто занадто великі, щоб усе рішення могло одночасно розміститись на єдиному чіпі. Це приводить до використання техніки часткової реконфігурації (partial dynamic reconfiguration, PDR) та спеціальної організації потоків. Часткова реконфігурація дозволяє модифікувати тільки частину конфігурації FPGA без зупинки виконання усієї задачі. Все більше пишеться багатопотокових програм і вони можуть бути значно прискорені на FPGA. Критичні ділянки коду можуть бути перенесені на FPGA та синтезовані прямо під час виконання програми.

Усі згадані вище підходи мають один недолік – відсутність системності. Жоден із них не охоплює процес розробки АЗ від задуму до реалізації. Є просто засоби організації мислення на зразок Y-chart, але вони не мають конкретних настанов для організації процесу розробки АЗ. Є концепція розробки АЗ на базі платформ, але всього-на-всього концепція, яка не включає у розгляд те, яким

чином побудувати складну систему, зазначивши тільки те, як можна збагатити базову платформу. Підхід із проблемно орієнтованими платформами також більше є концепцією, ніж повноцінним підходом до розробки, так само, як і TLM. Наразі галузі не вистачає системного та достатньо деталізованого підходу до розробки реконфігурованих систем. Існує велика кількість прийомів, концепцій, підходів, яка має перейти у якість.

У даному підрозділі наведено більш деталізований опис розробки АЗ з використанням АПС. У першому наближенні суть такого підходу до побудови апаратного та програмного забезпечень складається у відшукуванні прагматико-обумовленого компромісу між замкнутою та відкритою точками зору на них, який досягається за рахунок рекурсивного розподілу уявлень про об’єкт розробки на головні та другорядні. Саме такий процес підтримують АПС.

Як приклад розробки АЗ, до якого застосовні АПС, розглянуто розробки реконфігурованих систем. Проведено критичний огляд існуючих підходів і обґрунтовано доцільність застосування розробки з використанням АПС у цій галузі.

2.4 Композиційні засади адаптивних процесональних середовищ

Головною метою концепції АПС є реальна підтримка вирішення ІТ-задач. На загальному рівні досягнення цієї мети обумовлено викладеними вище принципами обумовленості, функціональності, експлікативності, композиційності та процесійності. На жаль, як було зазначено вище, більшість традиційних підходів дотримуються цих принципів лише номінально. Відтак підтримка ними вирішення ІТ-задач здійснюється, м’яко кажучи, лише «на словах», фактично зводячи її до забезпечення більш-менш зручної нотації отриманих рішень. Це є суттєвим обмеженням традиційних підходів. Зважаючи на фундаментальність причин такої обмеженості, ми розуміємо, що вона не може бути природним чином знівельована чи усунута. Змістовно кажучи, вона є принциповою складовою традиційних підходів. Саме тому для досягнення заявленої мети необхідним є створення нового інтеграційного підходу до

розробки (програмування, планування), який би, успадкувавши основні позитиви традиційних підходів, суттєво розвинув їх насамперед у в проблематиці, дослідження якої вимагає реального взаємодоповнення ІТП (програмування) та їхніх результатів. Це означає, що такий підхід не тільки на словах, а реально підтримував би взаємодоповнення основних аспектів розробки. Для цього він має володіти засобами підтримки прагматико-семантичних розглядів в ІТД. У створенні таких засобів центральне місце займають проективні функції (надалі – функції, якщо не обумовлене інше) та композиції.

2.4.1 ІТ-задача, функція та композиція в першому наближенні

Значущість функцій і композицій у ІТД на самому загальному рівні відзначена викладеними вище принципами функціональності, композиційності, експлікативності та процесійності. Тепер необхідно перейти від загальних констатацій корисності цих понять до їх поступового предметного збагачення. Для того, щоб не увійти в суперечність із принципами функціональності, композиційності та процесійності, збагачення функцій і композицій за необхідністю має бути відкрито-замкненим, тобто носити логіко-предметну природу. Адже, як вже зазначалося, саме логіко-предметне розуміння функції та композиції є змістовним носієм адаптивних процесональних середовищ. Зважаючи на принцип обумовленості, почнемо такі збагачення з композицій.

2.4.1.1 Основні властивості композицій

Говорячи про композиції як уточнення засобів генезису ІТ-рішення, насамперед виділимо їхні змістовні властивості, що пов’язані як із власне ІТД, так і з загальною, незалежною від ІТ-рішення природою композиції.

Однією зі специфічних рис ІТД як діяльності, що націлена на створення ІТ-рішень, є те, що у більшості випадків, окрім можливо найбільш простих, розробник як важливий суб’єкт ІТП не має не тільки формального опису завдання на розробку ІТ-рішення, а й навіть не володіє достатньо змістовним цілісним уявленням про нього. Вочевидь таке уявлення не може бути чимось

наперед даним, а навпаки — поступово генерується розробником відповідно до його суб'єктивного бачення того, як повинна вирішуватись поставлена перед ним задача в контексті відомої парадигми нерозривності зв'язку аналізу та синтезу (принцип «розділяй та володарюй»). Тобто вже на самому початку маємо справу насамперед із суб'єктно-обумовленим генезисом розуміння суті задачі як інтеграції розуміння сутей її підзадач. У цьому сенсі композиції як структури генерації ІТ-рішень, напевно, є фактично єдиними засобами змістовного суб'єктно-орієнтованого збагачення розроблюваного ІТ-рішення.

Передусім це вимога *адекватності* композицій змістовному розумінню розробником вирішуваної задачі. Інакше кажучи, композиції повинні відповідати тим уявленням про ІТД взагалі та про вирішення конкретної ІТ-задачі зокрема, якими володіє розробник. Якщо композиції не є адекватними, то розробник змушений трансформувати свої уявлення про вирішення задачі, щоб мати можливість викласти їх у термінах наявних генетичних структур. Саме такі трансформації є значним джерелом багатьох помилок у ІТД. Тому зрозуміло, що основним критерієм адекватності класу композицій є прагматико-обумовлена, тобто здійснювана суб'єктом, ІТД.

Властивість адекватності композицій безпосередньо пов'язана з розробником ІТ-рішення. Вона є одним із важливих чинників релятивізації ІТД. Однак необхідно також враховувати фактори впливу, які пов'язані власне з ІТД як діяльністю, шо націлена на створення ІТ-рішень. У цьому сенсі до композицій висувається вимога *замкненості*. Зміст її полягає в погляді на ІТ-рішення із точки зору сутності, стосовно якої коректно говорити про її суть – відповідну ІТ-функцію. У цьому сенсі будь-яке ІТ-рішення є носієм деякого сутесутнісного відношення (ССВ). ІТ-рішення і створюються саме задля тієї чи іншої імплементації його суті (його реалізації, виконання). Виконання ІТ-рішення природно передбачає наявність суб'єкту цього процесу. Саме тому властивість сутності «бути ІТ-рішенням» є релятивною. Таким чином, у першому наближенні *ІТ-рішення є сутністю, стосовно якої коректно говорити про деяку релятивну (суб'єктну) імплементацію її ІТ-функції.* Роль композиції у ІТД

відповідно до методу сутесутнісної релятивізації [18] полягає в об’єктивізації цієї релятивності в кожному конкретному ІТ-рішенні. Змістовно кажучи, композиція має *зберігати інформаційну технологічність* побудов, тобто бути замкнутою в ІТ-функціях. Це означає, що застосування композиції до ІТ-рішень як суб’єктно-залежних носіїв ССВ не виводить за клас таких ІТ-рішень[7]. Зі сказаного безпосередньо випливає *тотальність* композиції. Вона розуміється в тому сенсі, що коректно вказати на актуальний результат застосування композиції до будь-яких ІТ-рішень.

Властивості адекватності, замкненості та тотальності істотно обмежують клас композицій, однак він, як і раніше, залишається дуже загальним. Тому необхідне подальше внесення структур у клас композицій.

Структури композицій природно отримувати шляхом внесення до розгляду структур у класи ІТ-рішень, на яких задані композиції, а структури в класах ІТ-рішень визначати на основі структур даних. При цьому важливо підкреслити, що структури в класі ІТ-рішень і даних насправді індукуються структурами композицій, як це випливає з принципу обумовленості. Зважаючи на пріоритетність поняття композиції, перейдемо до розгляду структур композицій ІТ-рішень, пов'язаних зі структурами ІТ-функцій і даних.

2.4.1.2 Властивості ІТ-функцій та даних

Природно, що до розгляду повинні залучатися ті властивості функцій, які використовуються композиціями. Одну з них вже фактично закладено у визначенні композиції як *n*-арної операції. Представляючи вихідні функції у вигляді кортежу довжини *n*, ми отримуємо можливість розрізняти ці функції, а також виділяти аргументи та значення вихідних функцій за їхніми номерами (іменами).

Інша властивість функцій, що використовується композиціями, безпосередньо випливає з властивостей інформаційної технологічності та

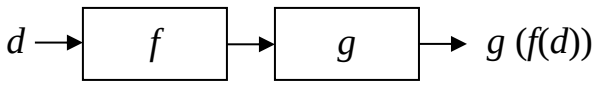
[7] Згідно з принципом функціональності, ІТ-рішення, з семантичної точки зору, є проективною функцією.

тотальності композицій. Вона полягає в тому, що передбачається можливість знаходження значення будь-якої вихідної функції на будь-якому зазначеному даному, якщо воно (значення функції) визначено. У розглянутих нижче прикладах композицій будуть використані тільки зазначені властивості функцій.

Щодо властивостей даних, то тут доцільно рухатись «від простого до складного». Спочатку природно з'ясувати, які класи композицій можуть бути отримані під час розгляду даних, до яких висувається мінімум вимог – абстрактних даних. При цьому елементи обраного класу даних, який будемо позначати D^A, розглядаються як «чорні ящики», в структуру яких не будемо поглиблюватись. Одержаний рівень розгляду назвемо абстрактним.

Абстрактні композиції. Розмаїття таких композицій є потенційно нескінченним, а сказаного про композицію взагалі поки що недостатньо для ґрунтовного збагачення згаданого розмаїття. Тому доцільним на цьому етапі є розглянути окремі репрезентативні приклади абстрактних композицій [12] [31] [90].

Найбільш вживаною абстрактною композицією є композиція множення $\cdot$. Змістовна її суть полягає в тому, що функціям f та g вона ставить у відповідність нову функцію $f \cdot g$ таку, що на будь-якому абстрактному даному $d \in D^A$ вона продукує значення $f \cdot g(d) \equiv g(f(d))$. Схематично це можна представити так:



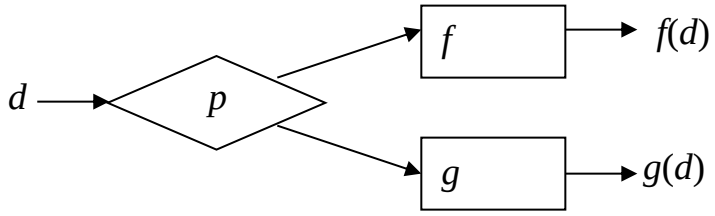
На цьому рівні можна презентувати цілий ряд композицій, однак практично всі вони тією чи іншою мірою будуть похідними композиції множення. Для принципового збагачення роду абстрактних композицій потрібне змістовне збагачення абстрактного типу даних D^A. Спочатку виділимо в D^A два спеціальні елементи T і F , які будемо інтерпретувати як «*true*» та «*false*» відповідно. Збагативши тип абстрактних даних множиною логічних значень

$\{T, F\}$, а ІТ-функції – функціями, що приймають логічні значення, тобто предикатами, можемо визначити ряд корисних абстрактних композицій.

Композиція галуження IF задається формулою

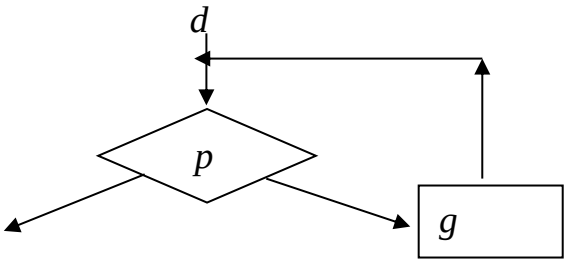
$$IF(p, f, g)(d) = \begin{cases} f(d), \textit{якщо } p(d) = T \\ g(d), \textit{якщо } p(d) = F \end{cases}$$

Схематично це виглядає так:



Композиція ітерації * функціям p і f ставить у відповідність функцію $*(p, f)$, значення якої на абстрактному даному $d \in D^A$ дорівнює першому з елементів послідовності абстрактному даному $d, f(d), f(f(d)),...$ для якого $p(d) = T$ за умови, що для всіх попередніх елементів послідовності значення $p(d)$ визначено. У системах алгоритмічних алгебр [91] аналогом композиції галуження є операція α-диз'юнкції, а композиції ітерації – операція α-ітерації.

Ця композиція може бути представлена наступною схемою:



Аналогічно до композиції галуження можна визначити композиції вибору $case\_of$ і деякі змістовні похідні від галуження та ітерації композиції.

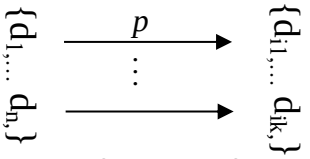
Таким чином, абстрактний рівень розгляду з виділеними елементами дозволяє визначити досить багатий клас композицій. Достатньо вказати на те, що

засоби, використовувані в системах алгоритмічних алгебр [92], у схемах Янова [93], у структурному програмуванні [15] [49] [50], можуть бути визначені на цьому рівні. Але зі змістовних міркувань зрозуміло, що клас композицій не обмежується тільки їхнім абстрактним різновидом. Адже у ІТД засоби генезису можуть істотно використовувати внутрішню структуру даних. Розуміння цього виводить розгляди за клас абстрактних композицій.

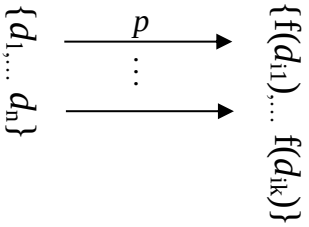
Множинні композиції. Мінімальним змістовним збагаченням розглядів є залучення до уваги на поряд із абстрактними даними множинного типу даних. Множинність дозволяє дане розглядати як скінчену множину, зокрема і множину абстрактних елементів. Позначимо цей тип даних, представниками якого є скінчені множини, як D^S.

Серед композицій множинного рівня відзначимо композиції виділення за умовою CS (Conditional selection) та композицію застосування за умовою CA (Conditinal application). Перша задається наступною формулою:

$CS(p)(s) \equiv \{d \mid d \in s \,\&\, p(d) = T\}$, де $s \in D^S$. Схематично це можна представити так:



Друга – формулою $CA(p, f)(s) \equiv \{f(d) \mid d \in s \,\&\, p(d) = T\}$ та схемою



Множинний тип загалом репрезентує досить багатий клас композицій [11] [33], проте обмежитися цим рівнем теж не можна. Це зрозуміло з природних міркувань. У ІТД засоби генезису можуть істотно використовувати не тільки внутрішньо-множинну структуру даних, але й більш складні структури. Зокрема розглядаються послідовні [94] та іменні структури даних [18] [12] [31] [32] [30]

[12] і відповідно n-арні та іменні функції та композиції, обумовлені такими структурами.

Кортежні композиції.

Щодо послідовних (кортежних) структур, то вони досить суттєво досліджені, наприклад у [18]. Там зроблено обґрунтований висновок про надмірну конкретність кортежних структур, що обтяжує розуміння функціональних і композиційних структур. Крім того, там обґрунтовано, що ІТД властиве використання іменних структур, зокрема структур типу скінчених множин іменованих елементів, так званих іменних множин [18] [30] [31] [32] [33]. Такими структурами добре моделюються зокрема кортежні структури. Тому доцільним є залучення іменних множин в якості нового типу абстракції даних. Цей тип абстракції розглядів назвемо іменним.

Іменні композиції.

Аналіз різних композицій ІТ-рішень показує, що вони використовують властивість даних бути множинами, що складаються з іменованих елементів. Такі множини називаються іменними. Даючи більш детальний опис, слід зазначити, що іменними вважаються скінчені множини виду $\{(v_1,a_1),(v_2,a_2),...,(v_k,a_k)\}\big|_{v_i\neq v_j,\forall i.j=1,...k}$, де . Тобто, виходячи з *принципу іменування* [11] [30] [31] [32] [33], іменні множини – скінченні функціональні бінарні відношення з $V\times D$, де V – множина імен, D – множина Значень (денотатів) у широкому розумінні.

Іменні множини дуже широко застосовуються в ІТД (і не тільки в ІТД). Репрезентативні приклади переваг застосування іменних структур перед кортежними ґрунтовно розглянуті, наприклад, у [18] [33] [94]. Тут йдеться про неадекватність використання так званого стандартного іменування в задачах розробки баз даних, обтяженість його в багатьох випадках специфікою послідовних структур даних та виграшність необтяженого послідовностями іменування, адекватність використання іменних множин для представлення

структур типу масивів, файлів, списків, таблиць, черг, стеків і т.п. Із наведеного випливає, що іменні множини є потужним інструментом специфікації важливих класів складених (структурованих) сутностей, що використовуються у ІТД. Такі структури можуть бути презентовані окремими видами іменних множин. Нами вже був відзначений зв'язок іменної множини з поняттями змінної та її значення. Інший приклад надає пам'ять обчислювальної машини. Тут адреси комірок виступають у якості імен, а вміст комірок – як значення імен.

Більш детальний огляд класів цих і подібних структур проведено, наприклад, у [18] [11]. Можна було б навести ще багато прикладів трактування важливих структур як іменних множин. Втім висвітленого достатньо, щоб переконатися в надзвичайній важливості імен, іменування та іменних множин у ІТД, програмуванні та їхніх спеціалізаціях. Аналогічні структури використовуються зокрема у Віденській мові визначень [95], скінченні функції (map) – у Віденській мові розробки [96], позиційні множини – у БД [97]. Представниками цих структур також є функціональні дані [14], комплекси [51], рядки значень [37] і т.д. Фактично це різні спеціалізації іменних множин.

Тепер перейдемо безпосередньо до іменного рівня розгляду. Він характеризується тим, що клас даних D збагачується виокремленням у ньому підкласу іменних множин. Нехай V і W – довільні множини, що тлумачяться відповідно як множини імен і значень. Причому допускається їх частковий непустий перетин та навіть співпадіння. Тобто можливо, що $V \cap W \neq \varnothing$. Клас D будується індуктивно. Вважаємо, що $W \subseteq D$. Інші елементи з D породжуються індуктивно за допомогою двох правил:

P1: якщо $v_1,...,v_k \in V$ попарно різні, а $d_1,...,d_k \in D$, то множина $\{(v_1,d_1),...,(v_k,d_k)\}$ також належить D $(k \geq 0)$;

P2: якщо $d_1,...,d_k \in D$, то множина $\{d_1,...,d_k\}$ також належить D $(k \geq 0)$

Виходячи з цього, адекватно трактувати будь-який елемент із D як іменне дане, або, якщо потрібно, явно підкреслити залежність класу даних D від V і W – (V,W) – іменне дане, а саму множину денотатів (значень, даних) D –

множиною іменних даних. Таким чином, до класу іменних даних потрапляють як дані з тривіальною іменною структурою (це насамперед всі елементи з W та скінченні підмножини з W), так і дані з більш складною, обумовленою застосуванням вищенаведеними правилами іменною структурою.

Домовимось під схемою іменної множини d розуміти скінченну множину імен $\{v_1,...,v_n\}$ – проекцію цієї множини за першою компонентою та позначати її $sh(d)$. Тобто $\{v_1,...,v_n\} = pr_1(d)$, де pr_i – функція проекції за i-ю компонентою m-арного відношення $(1 \le i \le m)$ [11].

Специфіка іменного рівня розгляду дозволяє нееклектично залучити до розглядів на відміну від абстрактного та послідовного рівнів, композиції не тільки над функціями, а й насамперед над іменними даними. Розглянемо репрезентативні приклади таких композицій [11] [30] [12] [31] [34] [35].

1. Композиція іменування $\to v$ (з параметром $v \in V$) іменує аргумент ім'ям v: $\to v(a) \equiv \{(v,a)\}, a \in D$

2. Операція розіменування $v \to$ (з параметром $v \in V$) вибирає з іменної множини значення імені v у цій множині:

$v \to (d) \cong a$, якщо $(v,a) \in d$ [8].

3. Операція накладення ∇ двом іменним множинам d_1 і d_2 ставить у відповідність нову іменну множину, що задається формулою

$d_1 \nabla d_2 \equiv d_2 \cup d_1^{"}$, де $d_1^{"}$ складається в точності з тих компонентів d_1, імена яких не входять в d_2. Тобто $d_1^{"} \subseteq d_1$ така, що $sh(d_1^{"}) \equiv sh(d_1) \setminus sh(d_2)$.

4. Операція додавання $\underset{i}{\cup}$ – звуження звичайного об'єднання множин на т.з. сумісні іменні множини: іменні множини d_1 і d_2 сумісні, якщо $d_1 \nabla d_2 = d_2 \nabla d_1$ (тобто для сумісних множин d_1 і d_2 однакові імена в їхніх схемах мають однакові значення). Тобто для сумісних множин d_1 і d_2: $d_1 \underset{i}{\cup} d_2 \equiv d_1 \cup d_2$.

---

[8] $\cong$ – означає узагальнену рівність

Тепер звернемо увагу на деякі важливі види іменних композицій над функціями. Почнем із композицій, що уточнюють засоби послідовного виконання функцій. Розглянемо простий приклад послідовного виконання операторів присвоювання: $v := v + u; u := v - u; v := v - u$, що міняє місцями значення змінних v та u.

Щодо функцій, які відповідають зазначеним операторам присвоювання, то є щонайменше два трактування: перша (абстрактна) точка зору полягає в тому, що оператори присвоювання переводять, у цілому, стани пам'яті в нові стани пам'яті, змінюючи значення навіть однієї зі змінних. Тому наведеним операторам присвоювання відповідають функції $f_1, f_2 . f_3$, які іменним множинам виду $\{(v,n),(u,m)\}$ ставлять у відповідність множини $\{(v,n+m),(u,m)\}$, $\{(v,n),(u,n-m)\}$ і $\{(v,n-m),(u,m)\}$. Друге трактування (іменне) проводить принципово іншу точку зору на організацію роботи операторів присвоювання. Змістовно кажучи, у цьому випадку наведені оператори присвоювання пов’язують із функціями $g_1, g_2 . g_3$, які вже більш суттєво враховують специфіку іменних даних та відзначену вище їхню близькість до структур пам’яті. Більш конкретно, ці функції множину $\{(v,n),(u,m)\}$ переводять відповідно в іменні множини $\{(v,n+m)\}$, $\{(u,n-m)$ та $\{(v,n-m)\}$.

Вочевидь, обидва трактування мають свої переваги та недоліки. Перше трактування зручне тим, що послідовне виконання операторів присвоювання може бути уточнене відносно простою композицією множення. Отже, семантика вищенаведеного терміну послідовного виконання операторів присвоювання задається функцією $f_1, f_2 . f_3$. Недолік трактування полягає в ускладненому розумінні семантики власне функцій $f_1, f_2 . f_3$, що ускладнює як їхні дослідження, так і дослідження засобів роботи з ними.

Іменне трактування явно вказує на зміни в пам’яті комп’ютера, що проводяться операторами присвоювання. У цьому сенсі воно краще підходить для дослідження функцій, які задають семантику програм. За такої умови

послідовне виконання буде задаватися вже не абстрактною композицією множення, а відносно більш складною за рахунок глибшого врахування специфіки роботи з пам'яттю, композицією, яку назвемо мультиплікуванням. Змістовна суть її на прикладі послідовного виконання двох операторів присвоювання полягає в наступному: на вихідних даних виконується перший оператор, отримані результати накладаються на вихідні дані, а потім виконується другий оператор. Таким чином, наслідком послідовного виконання операторів будуть результати виконання другого оператора, накладені на результати першого.

Даючи більш точний опис, слід зазначити, що операція мультиплікування – це бінарна операція, що двом іменним функціям h_1, h_2 ставить у відповідність нову іменну функцію $h_1; h_2$, значення якої на іменному даному d визначається так: $h_1; h_2(d) \cong d \nabla h_2(d \nabla h_1(d))$. Таким чином, у контексті іменного трактування послідовне виконання трьох вищезазначених операторів присвоювання може бути конкретизоване функцією $g_1; g_2; g_3$. Змістовно кажучи, ця трактовка є компромісом між відносно простішими та адекватнішими трактовками семантики операторів присвоювання, з одного боку, та більш складним у порівнянні з композицією множення, мультиплікуванням іменних функцій – з іншого. Складність останньої обумовлена тим, що ця композиція більш ґрунтовно враховує іменну специфіку даних і функцій, а значить – дозволяє більш ефективно використовувати апарат іменування в семантичних розглядах. Однак ця обставина свідчить скоріше не про недоліки іменної трактовки, а про адекватний суті речей «розподіл праці» між людиною та машиною у людино-машинній системі.

Композиція мультиплікування складає основу важливих іменних трактовок методів зациклювання, що вже знайшли своє відображення на абстрактному, множинному та кортежному рівнях. Так під іменною композицією циклування розуміємо бінарну операцію *WHD*, що іменному предикату $p: D \to \{T, F\}$ та іменній функції $f: D \to D$ ставить у відповідність

нову іменну функцію $WHD(p, f)$, значення якої на іменному даному d визначається так: $WHD(p, f)(d)$ покладається рівним першому з елементів послідовності $\varnothing, f(d), f; f(d),..., \underbrace{f;...; f}_{k}(d),...$ (нехай це буде $\underbrace{f;...; f}_{i}(d)$), для якого $p(\underbrace{f;....; f}_{i}(d)) = T$ за умови, що для всіх попередніх елементів послідовності значення предикату визначене та дорівнює F .

Аналогічно задається композиція галуження $IFTE$. А саме, $IFTE$ це тернарна операція, що іменному предикату $p: D \to \{T, F\}$ та іменним функціям $f_1: D \to D$ та $f_2: D \to D$ ставить у відповідність нову іменну функцію $IFTE(p, f_1, f_2)$, значення якої на іменному даному d визначається наступним чином: $IFTE(p, f_1, f_2)(d) \cong \begin{cases} f_1(d), \textit{якщо } p(d) = T \\ f_2(d), \textit{якщо } p(d) = F \end{cases}$.

Простежити відмінність абстрактної та іменної композицій галуження поки що видається несуттєвою. Вона краще простежується в композиції IFT , що уособлює досить популярний варіант галуження типу if_then . Вочевидь, композиція IFT може розглядатись як окремий випадок $IFTE$: $IFT(p, f) \cong IFTE(p, f, \Delta)$, для будь-яких функцій p, f . Тут Δ – так звана пуста іменна функція: $\Delta: D \to \{\varnothing\}$. Зазначимо, що в абстрактному випадку пуста функція задається дещо інакше: абстрактну функцію $e: D^A \to D^A$ таку, що $f(a) \equiv a$, для будь-якого абстрактного значення a будемо називати пустою абстрактною функцією. Виходячи з цього, зазначимо, що спеціалізація абстрактної операції галуження виду if_then задається наступним терміном: $IF(p, f, e)$.

Принципова відмінність між двома трактуваннями простежується на прикладі паралельного виконання операторів. Так паралельний оператор присвоювання, заданий, наприклад, у формі $v,u := u,v$, що фактично складається з двох операторів присвоювання $v := u$ та $u := v$, здійснює взаємний обмін значеннями змінних u,v. На абстрактному рівні важко визначити композицію,

яка формалізувала б паралельне виконання операторів. Водночас така композиція, яку будемо називати розпаралелюванням і позначати PRL, на іменному рівні може бути визначена як бінарна операція, що двом іменним функціям f_1, f_2 зіставляє нову іменну функцію $PRL(f_1, f_2)$, значення якої на іменному даному d визначається так: $PRL(f_1, f_2)(d) \cong f_1(d) \nabla f_2(d)$. Наприклад, покладемо, що f_1 задає семантику оператора присвоювання $v := u$, а f_2 – $u := v$, тобто $f_1(\{(v,n),(u,m)\}) = \{(v,m)\}$ та $f_1(\{(v,n),(u,m)\}) = \{(u,n)\}$. Тоді $PRL(f_1, f_2)(\{(v,n),(u,m)\}) \cong f_1(\{(v,n),(u,m)\}) \nabla f_2(\{(v,n),(u,m)\}) = \{(v,m),(u,n)\}$. Таким чином, паралельне виконання операторів дійсно взаємно міняє значення змінних v та u.

Виходячи з розглянутих прикладів, можна легко задати унарну композицію присвоювання $v :=$, що має параметр $v \in V$, поклавши $v := (f)(d) \cong \to v(f(d))$.

Композиції множення та мультиплікування задають семантику засобів послідовного виконання функцій. Увага до таких засобів обумовлена передусім специфікою абстрактного рівня розглядів. Однак іменний рівень забезпечує взаємодії, що принципово не могли бути розглянуті на абстрактному рівні та в дуже обмеженому вигляді – на кортежному рівні. У цьому контексті говориться про іменні суперпозиції.

Насамперед розглянемо суперпозицію за значенням. Цей спосіб взаємодії функцій зустрічається практично скрізь. Так, наприклад, простий вираз $v + u$ є результатом застосування суперпозиції за значенням функцій розіменування $v \to$ та $u \to$ у іменну функцію $+ : \{(1,m),(2,n)\} \to m + n$. Аналогічно n-арну функцію f^n будемо трактувати як іменну функцію $f : \{(1,a_1),(2,a_2),...,(n,a_n)\} \to f^n(a_1, a_2, ..., a_n)$. Тоді композиція суперпозиції S^{n+1} в іменну функцію f іменних функцій $g_1, g_2, ..., g_n$ задається як $n+1$-арна операція, значення якої на деякому іменному даному d задається формулою: $S^{n+1}(f, g_1, g_2, ..., g_n)(d) \cong f(\{(1, g_1(d)),(2, g_2(d)),...,(n, g_n(d))\}) = f^n(g_1(d), g_2(d), ..., g_n(d))$

Таким чином, семантика виразу $v+u$ може бути задана наступним чином: $S^3(+,v\to,u\to)(\{(v,n),(u,m)\})=+(n,m)=n+m$

Нехай тепер $f:D\to D$ - будь-яка іменна функція. Тоді суперпозицію можна проводити по довільним іменам $v_1,v_2,...,v_n$ і отримати наступне визначення загального випадку композиції суперпозиції за значенням $S^{v_1,v_2,...,v_n}$ – $n+1$-арну операцію, що $n+1$ іменній функції ставить у відповідність нову іменну функцію $S^{v_1,v_2,...,v_n}(f,g_1,...,g_n)$, значення якої на іменній множині d визначається наступною формулою: $S^{v_1,v_2,...,v_n}(f,g_1,...,g_n)(d)\cong f(d\nabla\{(v_1,g_1(d)),...,(v_n,g_{nb}(d))\})$.

Таким чином, іменний рівень розгляду дозволяє ввести у розгляди широко застосовувані засоби ІТД, в тому числі розробки АтаПЗ. Крім того, можна припустити, що композиції, які використовують властивості даних, визначаються на абстрактному, множинному та іменному рівнях. Ці властивості мають бути виділеним елементом, скінченою множиною чи іменною множиною. Це дозволяє стверджувати, що їхніх властивостей достатньо для визначення класу композицій ІТ-рішень. Тобто інші властивості даних, що використовуються композиціями, можуть бути зведені до розгляду перерахованих властивостей.

Зафіксуємо ці міркування у вигляді положення зводимості:

Положення зводимості. Для визначення композицій досить враховувати властивості функцій і даних, що розглядаються на абстрактному, множинному та іменному рівнях.

Звісно, ця теза не є формальним результатом і не може бути доведена у межах засобів формальної математики. Вона також не є догмою, що постулює лише одну точку зору на все розмаїття ІТ-задач, зокрема, розробки АтаПЗ. Цей результат лише фіксує сучасний стан речей у ІТД та програмуванні на основі аналізу ряду актуальних сьогодні репрезентативних методів вирішення задач. Попри всю важливість він не дає можливості реально просунутись у дослідженні ІТ-рішень і ІТД в цілому. Змістовно це не збагачує сформульовані вище засадничі тези обумовленості, підпорядкованості, відокремлюваності і

процесональності; не розкриває фундаментальної ролі композиції в такому збагаченні. Саме тому необхідне принципове збагачення композиційних розглядів. Проведені у цьому розділі дослідження та отримані результати складають реальні підстави для цього.

2.4.1.3 Прагматико-обумовлене збагачення композицій

Головним напрямком збагачення композицій є дослідження загального поняття композиції. Фундаментальне значення тут має викриття засадничих загальних властивостей композицій – *тотальності*, *адекватності* та *замкненості*. Ці властивості підтримують прагматичну обумовленість і релятивність виокремлення композицій як засобів розробки рішень серед різноманіття алгебраїчних операцій. Змістовно кажучи, саме вони об'єктивізують визначальні для прагматики задачі відношення між ключовими суб'єктами ІТ-рішення – його розробником і тим, на кого воно орієнтовано, тобто його користувачем. Так властивість адекватності забезпечує прагматичне обмеження композиції як алгебраїчної операції вимогою відповідності її змісту суб'єктивним уявленням про процес вирішення задачі. Властивість обчислюваності також є релятивним обмеженням композиції вимогою забезпечити можливість сприйняття одержаного рішення його користувачем. Таке трактування є суттєвим узагальненням традиційного розуміння замкнутості та природно зводиться до нього у тому разі, коли користувачем рішення є комп'ютер. Вимога тотальності фіксує той факт, що сьогодні відсутні прагматичні мотивації розгляду не всюди визначених засобів генезису рішень.

Викриті властивості суттєво збагачують клас композицій як алгебраїчних операцій. Попри це він все ще є надто загальним, а значить – недостатньо змістовним. Необхідним вбачається його подальше збагачення. Головним інструментом тут виступає прагматико-обумовлена типізація (ТОП). За допомогою ТОП у цьому підрозділі здійснено ряд важливих безпосередніх та опосередкованих збагачень поняття композиції, що покладені в основу прагматико-обумовленої класифікації всього класу композицій.

Одне з головних опосередкованих збагачень класу композицій, розглянуте тут, здійснено шляхом прагматико-обумовленої типізації сутностей, на яких можуть бути задані композиції. Зазначена вище орієнтованість цієї роботи на інформаційно-технологічну проблематику, зокрема на розробку програмно-апаратних рішень, суттєво обмежує клас таких сутностей. Це фактично звужує його до спеціальних класів функцій над носіями різних типів абстракції та рівнів загальності. Прагматико-обумовлена типізація цих класів є основним джерелом опосередкованих збагачень композицій.

Збагачення композицій природно отримувати шляхом ТОП класів функцій, на яких задані композиції, а збагачення в класі функцій – визначати на основі ТОП структур даних. При цьому важливо підкреслити, що збагачення в класі функцій і даних насправді індукуються структурами композицій, як це виходить із принципу обумовленості. На жаль, той факт, що в трійці понять "дане", "функція", "композиція" провідним є поняття композиції, часто не тільки не відзначається, але й не завжди усвідомлюється. Найбільш показово ця тенденція представлена у програмуванні. Тут вона простежується навіть у назвах багатьох методів програмування. Так у функціональному підході перевага віддається поняттю функції, у денотаційному методі акцент ставиться на дослідженні лише одного з засобів реалізації ІТД – оператора найменшої нерухомої точки, в модульному підході увага зосереджується на понятті модуля, тобто фактично на понятті функції, в методі програмування від даних (методі Джексона) основним виступає поняття даного. Проведені в даному підрозділі побудови у голову кута визначають пріоритетним поняття композиції, розглядаючи структури композицій у взаємодоповненні зі структурами функцій і даних.

Щодо функціональних властивостей, то тут інтерес представляють ті з них, що використовуються композиціями. Одна з властивостей фактично закладена у визначенні композиції як n -арної операції. Інша – у вимозі замкнутості композицій і полягає в тому, що передбачається можливість знаходження значення будь-якої вихідної функції на будь-якому даному, якщо це значення визначене.

Суттєвий вплив на залучення тих або інших властивостей даних здійснює інформаційно-технологічна спрямованість роботи. Аналіз різних композицій розробки АтаПЗ показує, що вони використовують властивості даних бути абстрактними даними, кортежами та множинами даних. При цьому можуть розглядатись кортежі та множини абстрактних, іменованих та метаіменованих елементів. Їх називають відповідно абстрактними, іменними та метаіменними кортежами і множинами.

Виходячи з наведеної ТОП даних та функцій, у роботі обґрунтовано сформовано первинну класифікацію композицій, що включає абстрактні, іменні та метаіменні класи композицій. Щодо кортежного типу, то дослідження даних, функцій та композицій цього типу показало, що кортежність функцій є надто обтяжливою в контексті синтезу таких функцій, а самі кортежні структури можуть бути легко змодельовані іменними множинами. Крім того, в роботі обґрунтована недоцільніс, якщо не сказати неможливість, вилучення з розгляду будь-якого з перерахованих типів композицій, окрім кортежного. Навіть за умови, якщо таке вилучення екстенсійно не вплине на потенціальну універсальність побудов, воно суттєво зашкодить інтенсійній адекватності розглядів.

У роботі проведене ґрунтовне дослідження класів абстрактних, іменних та метаіменних композицій, здійснена їхня подальша прагматико-обумовлена типізація. Зокрема розглянуто класи аплікативних і фундативних композицій як алгебраїчних операцій і їх абстрактні, іменні та метаіменні підкласи. Отримані в рамках цього дослідження результати безумовно мають самостійний інтерес. Однак у рамках цієї роботи вони важливі навіть не стільки самі по собі, скільки як такі, що забезпечили фактографічний фундамент для вирішення однієї з найважливіших задач роботи – дослідження загального поняття композиції.

2.4.2 Аплікативні композиції

У першому наближенні аплікативні композиції є засобами опосередкованого застосування засобів ІТД [98]. Вони розділяються на три типи: *класичні*, *неокласичні* та *некласичні* аплікативні композиції.

2.4.2.1 Класичні аплікативні композиції

Змістовно це такі засоби опосередкованого застосування засобів ІТД з одних об'єктів інших, в яких засоби ІТД та об'єкти розглядаються відповідно на гранично високому рівні абстракції.

Тепер розглянемо відповідно до наведеного інтуїтивного змісту композицій наступні уточнення. Універсум об'єктів O будемо тлумачити як універсальну множину, що складається з універсуму ІТ-рішень та універсуму предметів (композицій у пасивній ролі об'єкта), абстрактні об'єкти – як абстрактні елементи (дані) з універсальної множини O, а абстрактні засоби побудови – як абстрактні функції з O, які являють собою функції типу $f : O \to O$ (які можуть бути композиціями, наприклад).

Введемо далі класичну операцію, що отримала назву *аплікація*, яка кожній функції $f : O \to O$ та будь-якому елементу $d \in O$ ставить у відповідність новий елемент з O, який являє собою результат застосування функції f до аргументу d, який дорівнює значенню функції f на d та позначається $f(d)$. У разі, якщо воно не визначене, то і значення аплікації не визначене.

Не зважаючи на те, що прийняте тут технічно зручне позначення $f(d)$ для значення операції аплікації на парі (f, d) співпадає зі звичним позначенням значення функції f на d, це не приводить до плутанини, адже з контексту вимальовується предмет обговорення.

Безпосередньо з визначення аплікації випливає, що вона є адекватним уточненням усіх засобів опосередкованого застосування, які в принципі можливі на обраному гранично високому рівні абстракції. В силу принципів

композиційності, структурності та кластерності є всі підстави розглядати поняття аплікації як експлікацію поняття класичної аплікативної композиції.

Отримана таким чином експлікація інтуїтивного поняття застосування функції до аргументу у вигляді операції аплікації склала основу лямбда-числення, що виникло в основах математики та стало об'єктом пильної уваги теорії та практики програмування [66]. При цьому, як очікувалось, поряд із перевагами високого рівня абстракції це числення має принципові недоліки, які об'єктивно проявляються у спробах реального, а не номінального його використання в програмуванні.

Це все висунуло проблему пошуку таких аплікативних композицій, які зберегли б переваги класичних та мали значно менше недоліків останніх. У результаті були знайдені нові засоби опосередкованого застосування, що отримали назву неокласичних аплікативних композицій.

2.4.2.2 Неокласичні аплікативні композиції

Основна причина недоліків класичної аплікації ховається в тому, що найбільш суттєві для ІТД риси засобів опосередкованого застосування залишились поза згаданим уточненням її у вигляді аплікації. Щоб переконатись у справедливості цього твердження, достатньо аплікацію занурити в загальний контекст експлікації поняття засобів опосередкованого застосування.

Безпосередньо з визначення зрозуміло, що об'єкти (дані) розглядаються як "чорна скринька", в структуру якої не поглиблюємось. Навіть така властивість даних, як бути частиною інших даних, не розглядається.

Але ІТД, яка ігнорує цю властивість, перестає бути реальною ІТД. Адже його можна лише номінально зробити таким, щоб воно оперувало єдиним об'єктом. Для того, щоб розглядати цю фундаментальну властивість "бути частиною", на нього, як і на інші властивості, необхідно поглянути у розрізі логіко-орієнтованого стилю.

На перший погляд здається, що для появи принципово нових загально значимих (логічних), а не тільки суто специфічних рис, пов'язаних із цією

властивістю, достатньо знизити тип абстракції при розгляді неподільних об’єктів до об’єктів, що мають структуру множини. Адже тоді поняття частини об’єкту набуває природного сенсу. А саме, частина об’єкту як множини є підмножиною в звичайному теоретико-множинному розумінні.

Поняття “частина об’єкту” важливе не само собою, а через принципи композиційності та кластерності. Це призводить до необхідності забезпечити для засобів ІТД можливість їхнього застосовувати до об’єкту, використовуючи тільки його однозначно визначену частину, яка, можливо, співпадає з усім об’єктом. Втім на рівні засобів ІТД таке пониження абстракції нічого нового не несе.

Справа в тому, що дві різні підмножини, на яких застосовувана функція визначена, можуть бути частинами одного й того ж об’єкту. Тому без додаткової інформації однозначно вибрати частину об’єкту для використання його з метою вироблення результату застосування функції до цілого об’єкту неможливо.

Але додаткова інформація – вже специфіка, а не загальний випадок. Тому на рівні засобів ІТД нічого нового узагальнене застосування функції до об’єкту нести не може. Крім того, воно нічого не дає навіть на рівні загальних конструктивних засобів, тому що згадана специфіка носить принципово неконструктивну природу. Безпосередньо зі сказаного та принципів композиційності, структурності та процесійності виходить, що на цьому шляху в принципі не можуть бути отримані ані нові аплікативні композиції, ані, тим більше, фундативні. Таким чином, поява тут нової якості пов’язана з подальшим зниженням типу абстракції в розгляді об’єктів. Для того, щоб таке пониження мотивованим чином здійснити, необхідно ще раз звернутися до інтуїтивного змісту поняття засобу опосередкованого застосування.

Воно говорить про те, що об’єкти – не просто множини абстрактних елементів, а такі, що насамперед мають структуру, зокрема структуру іменованих елементів, яка передбачає наявність структури в універсумі об’єктів, що конкретизує його за рахунок індивідуалізації в ньому універсуму імен. В іншому разі ні про які більш глибокі застосування функції до об’єкту в

порівнянні з абстрактним випадком не можна говорити. Таким чином, множини іменованих елементів мають бути іменними множинами, або, більш точно, функціональними бінарними відношеннями між множинами імен та значень (денотатів).

Так можливо забезпечити застосування функцій до частин об'єктів. Засоби, які підтримують такий тип застосування, називаються аплікативними композиціями.

Нехай V – універсум імен, а U – скінченна підмножина V, тоді множину $d \subseteq V \times D$ домовимось називати іменною множину, якщо $\mathrm{Pr}_1(d) = U$, де Pr_k – функція проекції по k-ій компоненті, а $D \subseteq O$ – множина об'єктів, які інтерпретуються як дані. Тоді U-арна множина – множина U-іменних множин. Якщо завчасно відомо, яке U мається на увазі, таку множину називають просто *поліарною* множиною. У разі, якщо U складається з одного елементу, така множина називається *моноарною*.

Таким чином, будь-яка n-місна функція може розглядатись як U-арна функція з відповідним U. Для цього достатньо уявити випадок зі "стандартними іменами". Коли $U = \{1,...,n\}$, тоді фактично ім'я буде відповідати позиції елементу як аргументу n-місної функції. З використанням такого збагачення легко застосувати функцію лише до частини об'єкту, що суттєво розширює можливості класичної аплікації.

Під поліарною аплікацією будемо мати на увазі операцію, яка кожній поліарній функції f та будь-якій іменній множині $d \in O$, де O – іменний універсум об'єктів (даних) – ставить у відповідність новий об'єкт $f(\bar{d})$, де $\bar{d} \in Dom(f) \subseteq O$, $\bar{d} \subseteq d$ та $Dom(f)$ – область визначення функції f. У разі, якщо такого $\bar{d}$ не існує, або $f(\bar{d})$ не визначено, то і значення поліарної аплікації $f[d]$ як результат застосування функції f до об'єкту d не визначено. Легко побачити, що $\bar{d}$ визначається однозначним чином, тому все коректно.

Безпосередньо з визначення неокласичної аплікації витікає, що вона є адекватним уточненням згаданих засобів застосування функцій до об'єктів, які використовують тільки їхню частину. З цієї причини, а також в силу принципів програмологічності, композиційності, структурності та кластерності є всі підстави розглядати поняття неокласичної аплікації у якості експлікації поняття некласичної аплікативної композиції.

Не зважаючи на те, що поняття неокласичної аплікації значно багатше у порівнянні з поняттям класичної аплікації, воно відображає далеко не всі принципові риси засобів опосередкованого застосування функцій (засобів побудови) до об'єктів, що використовують лише їхні частини. Ця обставина висунула на перший план проблему пошуку нових аплікативних композицій, які були б повною мірою адекватними програмологічним засобам опосередкованого застосування.

У результаті були знайдені композиції, що йдуть принципово далі в порівнянні з неокласичними аплікативними композиціями. Вони явно залучають до розглядів парадигмові властивості засобів опосередкованого застосування, які були зовсім не характерні для класичних наук. Тому їх стали називати некласичними аплікативними композиціями.

2.4.2.3 Некласичні аплікативні композиції

Знизивши рівень абстракції до поліарних множин, ми наклали надто жорсткі умови на структуру застосовуваних функцій. Це не суперечить зазначеній вище загальнозначимості таких умов. Адже вони, по-перше, залишаються абсолютно не обтяжуючими для усієї математики і її традиційних застосувань. По-друге, поліарні функції як функції, для яких виконуються ці умови, у якості загальнозначимих структур добре поєднуються з доповнюючими їх специфічними структурами ІТД. Завдяки цьому вони широко та успішно використовуються в ньому.

Основний недолік згаданої жорсткості – складнощі роботи з іменами, що генеруються під час роботи готового ІТ-рішення. Зазвичай множина таких імен

потенційно необмежена. У традиційних дослідженнях, зокрема в математиці, це не було необхідним. Але в ІТД, особливо програмного забезпечення, генеровані імена (геноімена) досить природні.

Для того, щоб розглядати геноімена, необхідно насамперед за аналогією зі зниженням типу абстракції неподільних об'єктів абстрактного універсуму провести конкретизацію абстрактних іменних множин, знизивши тип абстракції у розгляді імен. Адже імена не є неподільними об'єктами у вигляді "чорних скриньок", природою яких не цікавляться у цьому разі.

Особливістю геноімен є те, що в жодній іменній множині не може бути використано більше одного значення функції, що генерує імена в якості імені. Інакше кажучи, в іменній множині у якості імені може фігурувати не більше одного представника (значення) функції, що генерує імена. Пов'язано це з тим, що в протилежному випадку узагальнене застосування функції до об'єкту було б пов'язане з нетривіальною специфікою – альтернативним вибором згенерованого значення. Це неможливо здійснити у рамках логічних засобів через принципи конструктивності та предикативності.

Іменні множини, для яких справедлива така генезисна особливість, іменуються геноіменними та визначаються наступним чином: нехай W – довільна, але фіксована множина, яка індивідуалізує метаімена у множині імен V, інакше кажучи, імена, денотатами яких є імена, що інтерпретуються як геноімена. Це множина, яка, згідно з Фреге [99] явно виділяє *опосередковані імена* поряд з *прямими іменами*.

Продовжуючи прийнятий термінологічний стиль, іменний універсум O з вказаною індивідуалізацією домовимось називати метаіменним універсумом або, враховуючи головну інтерпретацію денотатів метаімен, – геноіменним.

Під метаіменним універсумом будемо розуміти іменну множину d геноіменного універсуму O, яка задовольняє принцип метаіменування (геноіменування): яким би не було метаім'я, $\mathrm{Pr}_1(d)$ містить не більш ніж один його денотат.

Безпосередньо з визначення витікає, що, якщо $W = \varnothing$, то метаіменна (геноіменна) множина – просто іменна множина.

Геноімена індуктовані нетривіальними генетичними структурами. Крім вимоги до функцій “бути поліарною функцією”, вони мають вимогу, яка базується на також схематизації областей визначення функцій, що застосовуються до об’єктів. На відміну від розглянутого раніше випадку, вона має бути актуальною для не тільки для скінченних множин метаімен, але і для нескінченних множин геноімен, які можна інтерпретувати як області значень геноіменних функцій.

Для того, щоб конкретизувати її, позначимо через $U \subseteq V$ довільну скінченну множину імен, які не є метаіменами, а через $R \subseteq W$ – довільну скінченну множину метаімен. Тепер за аналогією з поняттям U – іменної множини, яка введена в рамках іменного універсуму, введемо поняття (U, R) – геноіменної множини в межах геноіменного універсуму O. З цією метою попередньо визначимо поняття омножинювання.

Під **омножинюванням** домовимось розуміти унарну операцію, яка зіставляє кожній сукупності елементів (не обов’язково різних) S множину усіх її різних елементів $\{S\}$.

Далі у прийнятих позначеннях за аналогією з поняттями прямого (декартового) множення та добутку введемо поняття прямого (декартового) додавання там суми.

Під **прямим (декартовим) додаванням** будемо розуміти n-місну операцію, яка кожній n-ці множин $(A_1,...,A_n)$ ставить у відповідність нову множину $\sum_{i=1}^{n} A_i = \{\{a_1,...,a_n\} / a_i \in A_i = \overline{1,n}\}$, яка називається прямою (декартовою) сумою.

У випадку, якщо $n = 2$, то $\sum_{i=1}^{2} \overset{def}{=} A_1 \oplus A_2$. На відміну від прямого множення, операція прямого додавання комутативна, так як

$A_1 \oplus A_2 = A_2 \oplus A_1$, але, як і перша, не асоціативна. Тому вона, так само, як і двомісна операція прямого множення, не може бути продовжена по асоціативності до n-місного декартового додавання.

Нехай далі $U_1,...,U_n$ – довільні, можливо, нескінченні множини імен, що інтерпретуються як множини денотатів метаімен із R, котрі э геноіменами, що генеруються зазвичай геноіменними функціями.

Метаіменну множину d будемо називати (U,R) - метаіменною множиною, якщо $\Pr_1(d) \in \{U \cup L / L \in \sum_{i=1}^{n} U_i\}$.

Використовуючи прийнятий термінологічний стиль, під (U,R)-арною множиною домовимось розуміти будь-яку множину (U,R)-метаіменних множин. У разі, якщо має значення те, що елементами множини є метаіменні множини та не важливо за яких саме U та R, такі (U,R)- ні множини домовимось іменувати просто метаполіарними множинами. В тому числі метаполіарні множини для яких $U = \varnothing$, а W складається з одного елементу, будемо називати **метамоноарними** множинами.

З визначення метаполіарних множин очевидно, що R виступає схемою нескінченної множини схем, які базуються як на попередньо заданих, так і на генезисних іменах. В цьому випадку вона (R) може розглядатись як **метасхема** по відношенню до цих схем.

Усе це дозволяє за аналогією з введеними раніше вимогами до функцій, що застосовують до об’єктів, адекватно конкретизувати нові програмологічні вимоги до них. Для того, щоб зробити це, насамперед домовимось за аналогією з U-арними функціями, іменувати функції, областями визначення яких є (U,R)-арні множини, (U,R) **-арними (метаполіарними, метамоноарними)** функціями.

При $R = \varnothing$ метаполіарні множини та метаполіарні функції вироджуються в поліарні множини та поліарні функції.

Виходячи з того, що клас метаполіарних функцій є адекватним розширенням класу поліарних функцій, на його основі можна дати уточнення програмологічного застосування функції до об'єкту.

За аналогією з поліарною аплікацією уточнення узагальненого програмологічного застосування назвемо метаполіарною аплікацією. Визначимо її формально.

Під **метаполіарною аплікацією** мається на увазі бінарна операція, яка кожній метаполіарній функції f та будь-якій метаіменній множині $d \in O$, де O – метаіменний універсум об'єктів (даних), зіставляє об'єкт $f(\bar{d})$, де $\bar{d} \in Dom(f)$ та $\bar{d} \subseteq d$. У випадку, якщо такого $\bar{d}$ не існує, або $f(\bar{d})$ не визначено, то значення метаполіарної аплікації $f\langle d \rangle$ як результат застосування функції f до об'єкту d не визначено.

Не важко помітити, що $\bar{d}$, як і у випадку поліарної аплікації, визначається наступним чином.

Відштовхнемося від протилежного: уявімо, що існує два різних d_1 та d_2 такі, що $d_1 \subseteq d$ та $d_2 \subseteq d$ та $d_1 \in Dom(f)$ і $d_2 \in Dom(f)$. Тоді $\Pr_1(d_1) \subseteq \Pr_1(d)$, $\Pr_1(d_2) \subseteq \Pr_1(d)$ та $\Pr_1(d_1), \Pr_1(d_2) \in \{U \cup L / L \in \sum_{i=1}^{n} U_i\}$. Але з $d_1 \subseteq d$, $d_2 \subseteq d$ та $d_1 \neq d_2$ слідує, що $\Pr_1(d_1) \neq \Pr_1(d_2)$ та $\Pr_1(d_1) \cup \Pr_1(d_2) \subseteq \Pr_1(d)$.

Значить деяка множина U_i $(i = \overline{1,n})$ представлена в d більш ніж одним своїм елементом, значить існує таке ім'я $r \in R$, що принаймні два його денотати (імені з U_i, що відповідає r) містяться в $\Pr_1(d)$. А це суперечить тому, що d є метаіменною множиною. Отже, $\bar{d}$ дійсно визначається однозначним чином, тому все коректно.

Спершу може здатися, що описані зниження типів абстракції можна продовжувати нескінченно, але це не так. Насправді процес отримання нових видів абстракцій обмежений лише трьома описаними видами: традиційною (абстрактною), неокласичною (поліарною) та некласичною (метаполіарною) аплікаціями.

Усе різноманіття типів абстракцій даних, як і будь-яке інше багатогранне поняття ІТД, має безтипову (універсумо типізовану) логіку та більш або менш багату специфіку, яка адекватно зводиться до типізації "по модулю" цієї логіки ІТД. Саме така тріада аплікацій складає основу аплікативної логіки ІТД, "по модулю" якої можуть бути адекватно типізовані всі аплікативні засоби ІТД.

В основі цього результату лежить наслідок прагматично повної системи принципів [98] та результатів про програмологічно повну типізацію даних [100] [13]. Суть цього наслідку в тому, що все різноманіття типів абстракції даних зводиться "по модулю" логіки ІТД до трьох типів (рівнів): абстрактний, іменний і метаіменний, які індуковані відповідними універсумами.

Конкретні прояви цього зведення проілюструємо у взаємодоповнюючому середовищі аплікативних композицій як засобів безпосереднього застосування.

2.4.3 Фундативні композиції

Природа фундативних композицій насамперед характеризується тим, що вони на якісно більш глибокому рівні порівняно з аплікативними композиціями відображають головну епістемологічну парадигму – "розділяй та володарюй". Тому вони ієрархічній композиційній структурі ІТД займають "верхні поверхи".

Одні композиції допускають експлікацію на всіх трьох рівнях абстракції, інші – можуть бути тільки на іменному. Групу, що існує на всіх рівнях абстракції, складають широко відомі засоби побудови ІТ-рішень як послідовне виконання, галудження та циклювання. Враховуючи таку загальність, розглянемо спершу їх.

2.4.3.1 Композиції типу послідовного виконання

Під послідовним виконанням інтуїтивно розуміється засіб побудови нового ІТ-рішення шляхом послідовного виконання ІТ-рішень-аргументів.

Цей засіб є бінарною алгебраїчною операцією, яка впорядкованій парі функцій (f,g) ставить у відповідність функцію fg (або $f \cdot g$), що називається *добутком* та задається наступною формулою $fg(a) = g(f(a))$.

На основі такого множення можна визначити операції піднесення до ступеня. Під n - ступенем будемо розуміти унарну операцію, яка кожній функції f ставить у відповідність нову функцію $f^n = (\underbrace{...(f \cdot f) \cdot f \cdot ... f}_{n\ раз})$. При цьому за визначенням f' вважаємо рівним тотожній функції E.

Інакше кажучи, множина усіх n - ступенів ($n = 0,1,2,...,n$) задається наступним індуктивним визначенням $f' = E$, а $f^{n+1} = f^n \cdot f$.

Множення, тим більше піднесення до ступеню, як джерело синтезу ІТ-рішень, взяті самі по собі, представляють лише обмежений інтерес. Адже ІТ-рішення як функції, як правило, не розкладаються на стільки сингулярні добутки простих функцій (ІТ-рішень).

Це все робить не актуальним пошук інших трактувань послідовного застосування окрім варіантів із функціями на абстрактних множинах даних. Більш багате трактування може бути отримане зі зниженням рівня абстракції до поліальних функцій.

Одне зі значно багатших трактувань одержимо, якщо, як і раніше, знизимо рівень абстракції в розгляді функцій до визначених раніше поліальних функцій.

Під (U,T)-альними функціями розуміються функції, областями визначення яких є U -арні множини, а областями значень — T -арні множини. У випадку, коли не важливо про які U та T йде мова, будемо, як раніше, іменувати їх **поліальними функціями**.

Отримане в прийнятих термінах уточнення послідовного виконання назвемо, враховуючи його зв’язок із множенням, мультиплікацією та формально визначимо наступним чином.

З цією метою попередньо введемо операції додавання та видалення.

Під **додаванням** мається на увазі бінарна операція $+$, яка кожній впорядкованій парі іменних множин d_1, d_2 ставить у відповідність нову іменну множину, що задається формулою $d_1 + d_2 = \bar{d}_1 \cup d_2$, де $\bar{d}_1$ складається з тих елементів, що не входять у d_2. Значення операції додавання будемо називати **сумою**.

Тепер домовимось параметричну операцію ext^U (від *extract*), яка кожній іменованій множині d ставить у відповідність іменну множину $ext^U(d)$, що отримується шляхом видалення з d іменованих елементів із іменами з U, називати U-видаленням.

Під **мультиплікацією** будемо розуміти бінарну операцію $\uparrow$, котра кожній впорядкованій парі (U_1,T_1) -, (U_2,T_2) -альних функцій f_1, f_2 зіставляє $(U_1 \cup (U_2 \setminus T_1), T_2 \cup (T_1 \setminus U_2))$-альну функцію $f_1 \uparrow f_2$, значення якої на $U_1 \cup (U_2 \setminus T_1)$ -іменній множині d задається в термінах операцій поліарної аплікації, додавання та видалення наступною формулою:

$$f_1 \uparrow f_2(d) = ext^{U_2} f_1[d] + f_2[d + f_1[d]].$$

У випадку, якщо значення якої-небудь операції, що фігурує у правій частинах формули, не визначено, то й значення $f_1 \uparrow f_2(d)$ не визначено.

У частковому випадку, коли f_1 та f_2 – (U,T)-альні функції, значенням операції мультиплікації знов буде (U,T)-альна функція, оскільки $U \cup (U \setminus T) = U, T \cup (T \setminus U) = T$. Значення ж $f_1 \uparrow f_2$ на U-іменні множині d задається більш простою формулою:

$$f_1 \uparrow f_2(d) = f_2[d + f_1[d]],$$

оскільки $ext^U f_1[d] + f_2[d + f_1[d]] = f_2[d + f_1[d]]$.

Коли ж $U = T$, то згадане значення задається більш простою формулою:

$f_1 \uparrow_2 (d) = f_2[f_1[d]]$, оскільки в цьому випадку $f_1[d]$ суть, як і d, U-іменна множина, тому $d + f_1[d] = f_1[d]$

Якщо ж $U \cap T = \varnothing$, то взагалі $f_1 \uparrow f_2(d) = f_2[d]$.

Зі сказаного витікає, що, якщо f є (U,T)-альною функцією, то $f_1 \uparrow f_2(d) = f[d + f[d]]$.

При цьому, якщо $U = T$, то $f_1 \uparrow f_2(d) = f[f[d]]$, а якщо $U \cap T = \varnothing$, то $f_1 \uparrow f_2(d) = f[d]$.

На цій основі звичайним чином визначаються операції піднесення до мультиплікативних степенів.

Під **мультиплікативним** n**-ступенем** мається на увазі унарна операція, яка кожні (U,T) -альній функції f зіставляє нову (U,T) -альну функцію $[f]^n = \underbrace{(...((f \uparrow f) \uparrow f)...)}_{n\ раз}, [f]^0 \overset{df}{=} E$.

Інакше кажучи, множина всіх мультиплікативних n-ступенів ($n = 0,1,2,\text{K}$) задається наступним індуктивним визначенням $[f]^0 = E$, а $[f]^{n+1} = [f]^n \uparrow f$.

Не дивлячись на те, що мультиплікація є далекосяжним неокласичним узагальненням множення, вона також не може розглядатись у якості експлікації послідовного виконання. Це витікає вже з того, що у ньому не знайшло ніякого відображення поняття геноімені, що, як вже згадувалось, визначає нетрадиційну сторону ІТД.

Перед тим, як дати некласичну експлікацію, введемо, як і раніше, допоміжні поняття.

Під **метадодаванням** будемо розуміти бінарну операцію ∇, котра кожній впорядкованій парі метаіменних множин d_1, d_2 зіставляє нову метаіменну множину, яка задається формулою $d_1 \nabla d_2 = \tilde{d}_1 \cup d_2$, де $\tilde{d}_1$ складається з тих елементів, імена яких не представлені ані в d_2, ані у вигляді абстрактних імен, денотатів метаімен, геноімен – денотатів одних і тих же метаімен. Значення операції метадодавання домовимось називати **метасумою**.

Домовимось далі двопараметричну операцію $ext^{U,R}$, котра кожній метаіменній множині d зіставляє метаіменну множину $ext^{U,R}(d)$, яка отримується шляхом видалення з d іменованих елементів із іменами U та геноіменами, які є денотатами метаімен з R, називати (U,R)-**видаленням**.

Метамультиплікацією називається бінарна операція, яка кожній впорядковані парі $((U_1,R_1),(T_1,Q_1))$–,$((U_2.R_2),(T_2,Q_2))$ -альних функцій f_1, f_2 ставить у відповідність $((U_1 \cup (U_2 \setminus T_1), R_1 \cup (R_2 \setminus Q_1)),(T_2 \cup (T_1 \setminus U_2), Q_2 \cup (Q_1 \setminus R_2))$-альну функцію $f_1; f_2$, значення якої на $(U_1 \cup (U_2 \setminus T_1), R_2 \cup (R_2 \setminus Q_1))$-метаіменній множині d задається наступною формулою:

$f_1; f_2(d) = ext^{U_2,R_2} f_1 < d > \nabla f_2 < d \nabla f_1 < d >>$.

У випадку, якщо значення якоїсь операції, що фігурує у правій частині формули, не визначено, то і значення $f_1; f_2(d)$ не визначено.

Безпосередньо з визначень місця знаходження, (U,R) -видалення та метамультиплікації слідує, що, якщо $R = \varnothing$, то вони перетворюються просто в додавання, U -видалення та мультиплікацію відповідно.

У випадку, коли f_1 та f_2 - $((U,R),(T,Q))$ -альні функції, значенням операції метамультиплікації знову буде $((U,R),(T,Q))$ -альна функція, адже $(U \cup (U \setminus T), R \cup (R \setminus Q)) = (U,R), T \cup (T \setminus U), Q \cup (Q \setminus R) = (T,Q)$. Значення ж $f_1; f_2$ на (U,R) -метаіменній множині d задається більш простою формулою: $f_1; f_2(d) = f_2 < d \nabla f_1 < d >>$, оскільки $ext^{U,R} f_1 < d > \nabla f_2 < d \nabla f_1 < d >>= f_2 < d \nabla f_1 < d >>$.

Коли ж $U = T$ та $R = Q$, згадане значення задається більш простою формулою:

$f_1; f_2(d) = f_2 < f_1 < d >>$, адже в цьому випадку $f_1 < d >$, як і d , є (U,R) -метаіменною множиною, тому $d \nabla f_1 < d >= f_1 < d >$.

Якщо ж $U \cap T = \varnothing$ та $R \cap Q = \varnothing$, то взагалі $f_1; f_2(d) = f_2 < d >$.

Зі всього сказаного випливає, що, якщо f суть $((U,R),(T,Q))$ -альна функція, то $f; f(d) = f < d \nabla f < d >>$.

На цій основі за аналогією до операцій піднесення до мультиплікативних ступенів вводяться операції піднесення до метамультиплікативних ступенів.

Під **метамультиплікативним n -ступенем** (піднесенням до метамультиплікативного n -ступеня) будемо розуміти унарну (параметричну) операцію, яка кожній $((U,R),(T,Q))$ -альній функції f ставить у відповідність нову $((U,R),(T,Q))$ -альну функцію

$$< f >^n = \underbrace{(...((f; f); f)...)}_{n\ раз}, < f > 0 \overset{df}{=} E .$$

Інакше кажучи, багато метмультиплікативних n -ступенів ($n = 0,1,2...$) задаються наступним індуктивним визначенням:

$< f >^0 = E$, а $< f >^{n+1} =< f >^n; f$.

Звернемось тепер до диз'юнкції (галудження) як загальному засобу синтезу ІТ-рішень, що допускає адекватні уточнення на всіх трьох рівнях абстракції – абстрактному, іменному та метаіменному.

2.4.3.2 Композиції типу ітерації

Інтуїтивно ясно, що суть будь-якого загальнозначимого (логічного) засобу типу ітерації (або циклювання) полягає у повторенні певної дії в межах заданої умови, яка перевіряється на початку наступного кроку повторення або після нього. У зв'язку з цими двома можливостями перевірки можна виділити два типи циклювань (ітерацій) – циклювання з постумовою та циклювання попередньою умовою.

Коли є поняття ступеню функції, мультиплікативного ступеню поліальної функції та метамультплікативного ступеню метаполіальної функції, не складно конкретизувати ці види циклювань на відповідних їм рівнях абстракції – абстрактному, іменному та метаіменному.

На абстрактному рівні, користуючись звичайним поняттям ступеню поліальної функції, не складно конретизувати ці види циклювань на відповідних їм рівнях абстракції – абстрактному, іменному та метаіменному.

На абстрактному рівні, керуючись звичайним поняттям ступеню функції, таке уточнення вже було визначено у згаданій роботі [101]. Воно було введене там у вигляді унарної параметричної операції – α-ітерації, де параметр α має раніше вказану суть. Нагадаємо її знову, дещо модифікувавши для зручності форму цього уточнення.

Під **абстрактною ітерацією (з попередньою умовою)** мається на увазі бінарна операція $*$, яка кожній впорядкованій парі p, f, де p - функція типу предикату, а f - довільна функція, ставить у відповідність нову функцію $p * f$ (чи $*(p, f)$), значення якої на будь-якому елементі $d \in O$ дорівнює першому з елементів послідовності $d, f(d),..., f^n(d),...$, на якому p приймає значення F, причому на всіх попередніх елементах значення p визначено та дорівнює T.

Аналогічним чином визначається операція **абстрактної ітерації (з постумовою)**, яку домовимось позначати символом $*$. Це не призведе до плутанини, оскільки з контексту завжди буде ясно, про що йде мова.

Само собою мається на увазі, що і класична абстрактна ітерація недостатньо розширює можливості побудови ІТ-рішень. Адже і вона, як бачимо, має справу з об'єктом як із "чорними скриньками", структура яких ніяк не враховується. Тому знизимо рівень абстракції до іменного.

Домовимось під **ітерацією (з попередньою умовою)** розуміти бінарну операцію $*$, котра кожній впорядкованій парі p, f, де p – $\overline{U}$ -арна функція типу предикату, а f - довільна (U,T) -альна функція, зіставляє $(U \cup \overline{U},T)$ -альну функцію $[p * f]$ (або $*[p, f]$), значення якої на будь-якому $U \cup \overline{U}$ - іменній множині d дорівнює першому з елементів послідовності $d, f[d],...,[f]^n[d],...$, поліарна аплікація якого з p приймає значення F, причому значення поліарної аплікації p з усіма попередніми елементами визначено та дорівнює T. Інакше кажучи, рівно n -тому елементу ($n = 0,1,2,...$) вказаної послідовності, що задовольняє ту саму умову, що і $p[[f]^n[d]] = F$, та $p[[f]^m[d]]$ визначено для всіх $m < n$ і дорівнює T.

Подібним чином вводиться операція **ітерації (з постумовою)**.

Не дивлячись на те, що неокласична ітерація значно розширює можливості абстрактної ітерації, вона також не є експлікацією інтуїтивного змісту циклювання як засобу синтезу ІТ-рішень. З причин аналогічних раніше вказаним, експлікуємо цей засіб у вигляді неокласичного метаіменного циклювання, яке домовимось називати просто метаітерацією.

Під метаітерацією (з постумовою) будемо розуміти бінарну операцію $*$, котра кожній впорядкованій парі p, f, де p - $(\overline{U},\overline{R})$ -арна функція типу предикату, а f - довільна $((U,R),(T,Q))$ -альна функція, ставить у відповідність $((U \cup \overline{U}, R \cup \overline{R}),(T,Q))$ -альну функцію $< p * f >$ (або $*< p, f >$), значення якої на будь-якому $(U \cup \overline{U}, R \cup \overline{R})$ -метаіменній множині d дорівнює першому з елементів послідовності $d, f < d >,...,< f >^n < d >,...$ метаполіарна аплікація якої з p приймає значення F, причому значення поліарної аплікації p з усіма попередніми елементами визначено та дорівнює T. Інакше кажучи, дорівнює n -

тому елементу ($n = 0,1,2,...$) вказаної послідовності, що задовольняє умову: $p << f >^{n} < d >>= F$ та $p << f >^{m} < d >>$ визначено для всіх $m < n$ і дорівнює T.

Аналогічно визначається операція **метаітерації (з постумовою)**.

Якщо з контексту буде ясно у відношенні до якого рівня абстракції розглядається ітерація, то слово “абстрактне” та префікс “мета”, як і в інших подібних випадках, домовимось опускати.

Інтегрована експлікаця інтуїтивних понять послідовного виконання, галудження, циклювання як засобів синтезу ІТ-рішень у вигляді (абстрактної, іменної та метаіменної) мультиплікації, диз’юнкції та ітерації може розглядатись у якості логіко-математичної засади структурного програмування у його традиційному розумінні. Однак, враховуючи те, що, на жаль, зазвичай такі засади звужують до уточнень на абстрактному рівні, варто особливо відзначити, що й інтегрованою експлікацією не можна обмежитись у реальній ІТД. Це справедливо хоча б тому, що при цьому експлікуються структури ІТД “макрорівня”, в той час, як в основі його лежать структури “мікрорівня”.

Особливе місце серед останніх займають структури так званого суперпозиційного типу. Вони розділяються на ряд видів і будуть вводитись за вмотивованою необхідністю.

2.4.3.3 Композиції типу суперпозицій

На відміну від загальних структур типу послідовного виконання, галудження та циклювання, вони не можуть бути уточнені, а, тим більше, експліковані на абстрактному рівні. Тому структури суперпозиційного типу навіть чисто номінально, подібно загальним структурам, не можуть бути адекватно втягнуті в ІТ-розгляди в межах логіко-математичних досліджень, які страждають зайвою традиційністю та залишають поза розглядом іменний і метаіменний рівні абстракції.

На інтуїтивному рівні структури суперпозиційного типу являють собою засоби побудови нових ІТ-рішень (функцій) із заданих шляхом підстановки результатів складових ІТ-рішень (функцій) в іншу функцію. При цьому ці підстановки можуть бути прямими, непрямими та змішаними.

Засоби суперпозиційного типу з прямими підстановками трактують значення допоміжних функцій як свої денотати. Тому їх на іменному рівні

називають денотатними суперпозиціями (суперпозиціями за значенням), на метаіменному – денотатними метасуперпозиціями. У випадку ж непрямих використань результатів ці значення трактуються як номінати (від лат. “імена”), по яким отримуються шукані денотати. Через це такі суперпозиційні засоби відповідно до рівня абстракції іменують номінативними суперпозиціями (мета суперпозиціями) чи супрепозиціями (метасуперпозиціями) за іменами. Якщо ж використання результатів змішане (як у якості імен, так і у якості денотатів), такі засоби називаються просто суперпозиціями (метасуперпозиціями).

Перш за все, конкретизуємо найбільш прості з цих засобів – денотативні суперпозиції.

Нехай f - довільна U -арна функція, g_i - які-небудь U_i -арні функції ($i=\overline{1,n}$), а $u_1,...,u_n$ – послідовність різних імен з U.

Під денотатною суперпозицією будемо розуміти $n+1$ -арну (параметричну) операцію $SD^{u_1,...u_n}$, котра кожному кортежу $f,g_1,...,g_n$ функцій згаданого типу зіставляє нову $U\setminus\{u_1,...,u_n\}\cup\bigcup_{i=1}^{n}U_i$ -арну функцію $SD^{u_i,...u_n}(f,g_1,...,g_n)$, яка задається наступною формулою:

$$SD^{u_1,...u_n}(f,g_1,...,g_n)(d)=f[d+\{(u_1,g_1[d]),...,(u_n,g_n[d])\}].$$

Домовимось функції, значеннями яких є номінати з універсуму номінат (імен) V, називати **номінатозначними функціями**.

Нехай тепер ν - довільне ім’я з універсуму імен V, а d – іменна множина з іменного (метаіменного) універсуму O. Тоді під функцією вибору Sel будемо розуміти двомісну функцію, яка задається формулою

$$Sel(\nu,d)=\begin{cases}\sigma, \textit{ якщо } (\nu,\sigma)\in d\\ \textit{не визначено}\end{cases}$$

Уявімо далі, що f та $u_1,...,u_n$ мають попередній сенс, g_i – довільні U_i -арні номінатозначні функції, а $R=\{r_i,...,r_n\}$ - множина метаімен така, що множина денотатів кожного r_i в точності дорівнює множині значень функції g_i :

$Ran(g_i)=\{g_i(a)/a\in Dom(g_i)\}$ ($i=\overline{1,n}$). Тоді в прийнятих позначення поняття номінатної суперпозиції визначається наступним чином.

Номінатною суперпозицією будемо називати $n+1$-арну (параметричну) операцію $SN^{u_1,...u_n}$, котра кожному кортежу $f,g_1,...,g_n$ функцій згаданого типу ставить у відповідність $((U\setminus\{u_1,...,u_n\})\cup\bigcup_{i=1}^{n}U_i,R)$ -арну функцію $SN^{u_1,...u_n}(f,g_1,...,g_n)$, яка задається наступною формулою:

$SN^{u_1,...u_n}(f,g_1,...,g_n)(d)=f[d+\{(u_1,Sel(g_1[d],d)),...,(u_n,Sel(g_n[d],d))\}]$.

Нехай f_i ($i=\overline{1,m}$) – довільні T_i-арні функції, а $t_1,....,t_m$ - послідовність різних імен з U, які відмінні від $u_1,...,u_n$. Тоді, зберігши за іншими позначеннями попередній сенс, введемо більш загальне поняття суперпозиції.

Суперпозицією є $m+n+1$-арна (параметрична) операція $S^{t_1,...t_m;u_1,...u_n}$, яка кожному кортежу $f,f_1,...,f_m,g_1,...,g_n$ функцій згаданого типу зіставляє нову $((U\setminus(\{t_1,...,t_m\}\cup\{u_1,...,u_n\}))\cup\bigcup_{i=1}^{m}T_i\cup\bigcup_{i=1}^{n}U_i,R)$ -арну функцію $S^{t_1,...t_m;u_1,...u_n}(f,f_1,...,f_m,g_1,...,g_n)$, яка задається формулою

$S^{t_1,...t_m;u_1,...u_n}(f,f_1,...,f_m,g_1,...,g_n)(d)=$

$f[d+\{(t_1,f_1[d]),...,(t_n,f_n[d]),(u_1,Sel(g_1[d],d)),...,(u_n,Sel(g_n[d],d))\}]$

Введені суперпозиції в своїй основі можуть розглядатись як адекватні уточнення засобів суперпозиційного типу на іменному рівні. Однак за причинами, які ми неодноразово вказувати, вони не є експлікацією таких засобів. Тому за необхідністю варто знов знизити об'єктний рівень абстракції до метаіменного рівня та розглядати поряд із поліарними метаполіарні функції. Більше того, на відміну попередніх випадків клас поліарних функцій, як слідує з визначень, не замкнутий відносно номінатних та загальних суперпозицій.

Нехай тепер f – довільна (U,R)-арна функція, g_i - якісь (U_i,R_i)-арні функції ($i=\overline{1,n}$), а $u_1,...,u_n$ - послідовність різних імен з U.

Домовимось під денотатною метасуперпозицією розуміти $n+1$ -арну (параметричну) операцію $SMD^{u_1,...u_n}$, котра кожному кортежу $f,g_1,...,g_n$ функцій

згаданого типу ставить у відповідність $((U \setminus \{u_1,...,u_n\}) \cup \bigcup_{i=1}^{n} U_i, R \cup \bigcup_{i=1}^{n} R_i)$ -арну функцію $SMD^{u_1,...u_n}(f, g_1,..., g_n)$, що задається наступною формулою:

$$SMD^{u_1,...u_n}(f, g_1,..., g_n)(d) = f < d \nabla \{u_1, g_1 < d >),...,(u_n, g_n < d >)\} >.$$

Тепер припустимо, що g_i - довільні (U_i, R_i)-арні номінатозначні функції, а $Q_i = \{q_1,..., q_n\}$ – множина метаімен, множинами денотатів яких відповідно є $Ran(g_i)$ ($i = \overline{1,n}$). Тоді в прийнятих позначеннях поняття номінатної метасуперпозиції вводиться наступним чином.

Номінатною метасуперпозицією вважається $n+1$-арна (параметрична) операція $SMN^{u_1,...u_n}$, яка кожному кортежу $f, g_1,..., g_n$ згаданого типу зіставляє $((U \setminus \{u_1,...,u_n\}) \cup \bigcup_{i=1}^{n} U_i, Q \cup \bigcup_{i=1}^{n} R_i)$ -арну функцію $SMN^{u_1,...u_n}(f, g_1,..., g_n)$, яка задається наступною формулою:

$$SMN^{u_1,...u_n}(f, g_1,..., g_n)(d) = f < d \nabla \{(u_1, Sel(g_1 < d >, d)),...,(u_n, Sel(g_n < d >, d))\} >.$$

Тепер позначимо через f_i ($i = \overline{1,m}$) – довільні (T_i, Q_i) -арні функції, зберігши попередній (останній) сенс за іншими позначеннями.

У прийнятих позначеннях загальне поняття метасуперпозиції експлікується наступним чином.

Метасуперпозицією називається $m+n+1$-арна (параметрична) операція $SM^{t_1,...t_m;u_1,...u_n}$, яка кожному кортежу $f, f_1,..., f_m, g_1,..., g_n$ функцій вказаного типу ставить у відповідність $((U \setminus (\{t_1,...,t_m\} \cup \{u_1,...,u_n\})) \cup \bigcup_{i=1}^{m} T_i \cup \bigcup_{i=1}^{n} U_i, Q \cup \bigcup_{i=1}^{m} Q_i \cup \bigcup_{i=1}^{n} R_i)$ -арну функцію $SM^{t_1,...t_m;u_1,...u_n}(f, f_1,..., f_m, g_1,..., g_n)$, що задається наступною формулою:

$$SM^{t_1,...t_m;u_1,...u_n}(f, f_1,..., f_m, g_1,..., g_n)(d) =$$

$$f < d \nabla \{(t_1, f_1 < d >),...,(t_m, f_m < d >),(u_1, Sel(g_1 < d >, d)),...,(u_n, Sel(g_n < d >, d))\} >.$$

Безпосередньо з визначень денотатної метасуперпозиції, номінатної метасуперпозиції та метасуперпозиції витікає, що на поліарних функціях вони збігаються з відповідними суперпозиціями. Варто зазначити, що

метасуперпозиції у класі метаполіарних функцій є алгебраїчними операціями, аналогічними суперпозиціям у класі поліарних функцій.

Алгебраїчність метасуперпозицій, як і інших композицій, має принципове значення. Вона закладена в основі покрокової побудови ІТ-рішень, бо адекватно підтримується інтерфейс між окремими кроками ІТД. Розкриття природи згаданого інтерфейсу є важливою задачею у межах розбудови концепції АПС. Вирішення її – багатоланковий процес. Але ж починати його треба з «початку» – відповіді на питання про існування такого рішення. Денотативна парадигма тут грає непересічну роль.

У цьому розділі викладено теоретичні основи АПС. Тут роз'яснюється природа нового відкрито-замкнутого підходу до вирішення сучасних ІТ-задач. А саме, спираючись на принципи функціональності, композиційності та процесійності обґрунтовується залучення до розглядів поняття композиції у якості основного засобу для роз'яснення процесу побудови ІТ-рішень. Цим визначається шлях до нівелювання основних недоліків традиційних методів ІТД – домінування результату в ІТД, протиставлення підходів до ІТД один одному, нездатність адекватно враховувати активну роль суб'єкта в ІТД та її вплив як на ІТП, так і на їхні наслідки. Ці питання ставляться та вирішуються шляхом введення та дослідження нового поняття – КСВ. Це обумовлена прагматикою цієї роботи конкретизація сутесутнісного відношення. Поняття КСВ дозволило принципово збагатити конкретикою принцип процесійності через розуміння ІТД як замикання КСВ. А те, що КСВ – частковий порядок, дозволило залучити до розглядів КСВ традиційний математичний апарат і отримати результати, що не тільки збагатили власне КСВ, але і дали можливість суттєво просунутись у дослідженні загального поняття композиції. Дослідження загального поняття композиції є ключовим у розділі. Фундаментальне значення тут мають засадничі загальні властивості композицій – тотальність, адекватність та замкнутість. Ці властивості підтримують прагматичну обумовленість і релятивність виокремлення композицій як засобів розробки ІТ-рішень серед різноманіття алгебраїчних операцій. Зазначені властивості суттєво збагачують клас

композицій як алгебраїчних операцій. З метою подальшого збагачення поняття композиції, у розділі проведена ТОП композицій.

Аналіз різних композицій апаратно-програмного ІТД показує, що вони використовують властивості даних бути абстрактними даними, кортежами та множинами даних. При цьому можуть розглядатись кортежі та множини абстрактних, іменованих і метаіменованих елементів. Їх називають відповідно абстрактними, іменними та метаіменними кортежами та множинами.

Виходячи з наведеної ТОП даних і функцій, у розділі сформовано первинну класифікацію композицій, що включає абстрактні, іменні та метаіменні класи композицій. Отримані в межах класифікації результати безумовно мають самостійний інтерес. Однак у цій роботі вони важливі навіть не стільки самі собою, скільки як такі, що забезпечили фактографічний фундамент для вирішення однієї з найважливіших задач роботи – дослідження загального поняття композиції.

Перехід до відкрито-замкнутих розглядів, адекватно відображених у композитосутнісних відношеннях, дозволив не тільки усунути виявлені неадекватності згаданих традиційних методів, але й інтегрувати їх із методами, що розвиваються в межах денотативних підходів. У цьому розділі окремі результати денотаційного підходу були спроектовані на уведені вище КСВ.
Оскільки функції, індуковані реальною ІТД, складно влаштовані, підвести під них єдину денотативну базу неможливо, але можливо дати реальну основу для індивідуалізації необхідної функції серед множини таких функцій. Безпосередньо з вищенаведеної прагматико-обумовленої аргументації визначення композитосутнісних відношень випливає, що вони можуть розглядатися в якості такої основи. Принаймні вони дозволяють індивідуалізувати типи вищезгаданих ІТ-функцій. У розділі доведена теорема про нерухому точку ІТ-функцій. Принципова значимість цієї теореми полягає в тому, що вона відкриває зовсім природній шлях для дослідження тих типів прагматико-обумовлених функцій, для яких їхнє існування є більш-менш прямим наслідком такої обумовленості. Такий підхід є не просто відмінним від

відомих традиційних, а й дуальним до них, не конкуруючим, а істотно взаємодоповнюючим. Отримані тут результати складають логічну основу адаптивних процесональних середовищ, а наступні розділи дисертації по суті присвячені одному – наповненню цієї основи предметним змістом.

У розділі досліджуються класи композицій, що пов’язані з ІТД, зокрема розробкою АтаПЗ. Вони складають предметну основу адаптивного процесонального середовища. Останнє є застосуванням універсального середовища інтеграції [18] до задач розробки апаратного та програмного забезпечень. Наведена прагматико-обумовлена типізація композицій, що логіко-предметно підтримують ІТД. Сформульовано теорему про нерухому точку проективної функції та викладено композитосутнісний метод розробки АтаПЗ.

РОЗДІЛ 3. СЕМАНТИЧНІ ДОСЛІДЖЕННЯ АПС

Після формування концепції АПС, цілком природньо постає проблема розробки мов проектування для відчуження результатів проектування, що в свою чергу диктує необхідність відповідних семантичних досліджень.

Проблема еталонування семантики проектів як формального її опису виникла з появою програмування [100] [13]. Особливо актуальною вона стала у наш час. Сьогодні все частіше говорять про кризу проектування, маючи на увазі труднощі застосування науково обґрунтованих методів синтезу проектів у повсякденній практиці проектувальників.

Пошук науково обґрунтованих підходів до розгляду проектів ведеться в різних напрямках. Великі надії покладаються на розвиток алгебраїчних методів дослідження проектів. В основі таких методів лежать програмні алгебри – алгебри, носіями яких є спеціальні класи функцій, а операціями – композиції, що являють собою абстракції від засобів синтезу проектів. Зокрема, в термінах цих алгебр точно ставляться і вирішуються проблеми повноти в різних класах обчислюваних функцій, які (проблеми) займають одне з важливих місць у розробці програмного та апаратного забезпечення.

3.1 ППА в класі функцій над записами, що не зберігають денотати

3.1.1 Загальні положення

Носій ППА складають n-арні функції та n-арні предикати (далі – функції та предикати) ($n = 1, 2, \ldots$). Сигнатуру же ППА (що позначається тут як Ω) складають операції суперпозиції, розгалудження та циклювання, що являють собою адекватні уточнення основних методів конструювання програм, які властиві більшості високорівневих мов програмування [102] [103]. Нагадаємо формальні визначення цих операцій. Для зручності та компактності викладення при позначенні функцій і предикатів перевага буде надаватись не операторній, а термальній формі запису [67].

Нехай задані m функцій $f_1,...,f_m$ однакової арності (наприклад, k) вигляду $A^k \to B$, визначені на деякій попередньо зафіксованій множині A зі значеннями з множини B [67] [103] (не обов'язково, щоб $A \cap B = \varnothing$; допускається ситуація, коли $A \cap B = \varnothing$) та нехай на множині B визначена m-арна функція f, що приймає значення з деякої множини C. Розглянемо нову k-арну функцію $g : A \to C$, таку, що її значення на аргументі $< a_1,...,a_k >$ задається наступним чином $g(< a_1,...,a_k >) \cong f(f_1(< a_1,...,a_k >),..., f_m(< a_1,...,a_k >)))$. Будемо вважати, що функція g є результатом застосування $(m+1)$-арної *суперпозиції*, що позначається як S^{m+1}, до кортежу функцій $< f, f_1,..., f_m >$, інакше кажучи, $g \equiv S^{m+1}(< f, f_1,..., f_m >)$.

Нехай тепер крім вже даних функцій $f_1,..., f_m$ додатково задані функція h того ж вигляду $A^k \to B$ та m-значна функція $\beta : B \to \{1,2,...,m\}$. Будемо вважати, що k-арна функція $g : A^k \to B$ виникає з функцій $h, f_1, \ldots, f_m$ $(m+1)$-арної параметричної операції розгалуження $\Diamond_\beta^{m+1}$, якщо для довільного аргументу $< a_1,...,a_k > \in A^k$ значення функції $g(< a_1,...,a_k >)$ задається наступним чином: $g(< a_1,...,a_k >) \cong f_r(< a_1,...,a_k >)$, якщо $\beta(h(< a_1,...,a_k >)) = r$, $(1 \le r \le m)$.

Відмітимо, що таким чином введена параметрична операція розгалуження являє собою адекватне уточнення відомого методу конструювання програм типу *case_of*. Корисним частковим випадком цієї операції вочевидь є тернарна операція *розгалуження* $\Diamond$, яка ставить у відповідність двом функціям $f_i : A^k \to B,\ i = 1,2$ та одному предикату $p : A^k \to \{T, F\}$ k-арну функцію $g \equiv \Diamond(< p, f_1, f_2 >)$, значення якої на довільному аргументі $< a_1,...,a_k > \in A^k$ задається так:

$$g(< a_1,...,a_k >) \cong \begin{cases} f_1(< a_1,...,a_k >), p(< a_1,...,a_k >) = T \\ f_2(< a_1,...,a_k >), p(< a_1,...,a_k >) = F \end{cases}$$

Насамкінець поповнимо список заданого k-арним предикатом $p: A^k \to \{T, F\}$. Розглянемо k-арну функцію $g: A^k \to B$, значення якої $g(< a_1,...,a_k >)$ на довільному аргументі $< a_1,...,a_k > \in A^k$ вважається рівним першій компоненті першого кортежу послідовності кортежів $[< a_1^i,...,a_k^i >]_{i=0,1,2,...}$, де $a_j^0 = a_j$, $j = 1,2,...,k$ та $a_j^{i+1} = f_j(\langle a_1^i,...,a_k^i \rangle)$, $j = 1,2,...,k$, для якого (позначимо його, наприклад, $< a_1^s,...,a_k^s >$) $p(< a_1^s,...,a_k^s >) = F$ за умови, що для всіх $r = 1,2,...,s-1$ значення $p(< a_1^r,...,a_k^r >) = T$. Функція g виникає застосуванням $(m+1)$-операції *циклювання* до функцій кортежу $< p, f_1,\ldots, f_m >$. Домовимось позначати її $g \equiv *^{m+1}(< p, f_1,\ldots, f_m >)$. Відповідно до цього $g(< a_1,...,a_k >) = a_1^s$. Відзначимо, що досі для позначення введених операцій нами використовувалась виключно операторна форма запису. Відтак, використовуючи термальну форму запису операцій сигнатури ППА, будемо явно вказувати тільки ті змінні, значення котрих мають вагу в цьому випадку. Наприклад, для операції циклювання будемо використовувати такий запис: $p(x_1,\ldots,x_m) \underset{y_1\ldots y_k}{*} < f_1(z_1^1,\ldots,z_{m_1}^1),\ldots, f_m(z_1^k,\ldots,z_{m_k}^k) >$, вказуючи явно тільки ті змінні, від яких функції та предикат значно залежать. При цьому функція $f_j(z_1^j,\ldots,z_{m_j}^j)$ «керує» змінною y_j, $j = 1,2,...,k$, а змінна y_1 вважається «вихідною». Спосіб відновлення операторної форми тут очевидний.

Зафіксуємо тепер деяку зліченну множину , а для деякого $k \in N$ розглянемо класи Φ^k часткових k-арних функцій і предикатів вигляду $D^k \to D$ та $D^k \to \{N, F\}$ відповідно та $\Phi \equiv \bigcup_k \Phi^k$ багатомісних часткових функцій і предикатів на D. Далі під функціями (предикатами) на D, D-функціями (D-предикатами) будемо розуміти функції (предикати) з Φ. Обчислюваність на D вводиться як нумераційна обчислюваність [104]. Домовимось через $A_D^{чр}$ позначати ППА, носій якої складають частково-рекурсивні функції

(обчислювані функції, чр-функції) на D. Породжуючу множину алгебри $A_D^{чр}$, назвемо її *повною системою* (ПС), а повну систему ППА – її I_m^n*-базисом*, якщо будь-яка її підсистема, що отримується шляхом видалення з неї будь-якої функції, відмінної від селекторної чи предиката, вже не буде повною.

При дослідженні повних систем ППА корисними будуть деякі наведені нижче позначення, терміни, а також властивості та супутні їм результати.

Властивість. n-арна функція f зберігає множину $L \subset D, L \neq \varnothing$, якщо $f(\underbrace{L \times \ldots \times L}_{n}) \subseteq L$ [103].

Нехай тепер D – множина обє'ктів, відносно яких коректно говорити про їхні складові. Уявімо, що універсально множина таких складових є зліченною. Позначимо її через B. Зафіксуємо відображення $\beta : D \to 2^B$, де 2^B — множина всіх скінченних підмножин множини B. Змістовно кажучи, для будь-якого $d \in D$ $\beta(d) \in 2^B$ – множина елементів із B, які є складовими d — його денотатами.

Властивість. n-арна функція f β-зберігає денотати, якщо існує скінченна множина $B_f \subset B$, така, що для будь-якого $d \equiv < d_1, ..., d_n > \in \mathrm{dom}\, f$ справедливо

$$\beta(f(< d_1, ..., d_n >)) \subseteq \bigcup_{i=1}^{n} \beta(d_i) \cup B_f.$$

Цікаво те, що наведені властивості D-функцій зберігаються в сигнатурі Ω. Це дозволяє сформулювати ряд простих та корисних необхідних умов повноти для ПС $A_D^{чр}$.

Твердження. Будь-яка повна система алгебри $A_D^{чр}$ для будь-якої непустої множини L $(L \subset D,\ L \neq \varnothing)$ містить D-функцію, котра не зберігає множину L.

Твердження. Будь-яка повна система алгебри $A_D^{чр}$ містить хоча б одну D-функцію, котра не β-зберігає денотати.

Корисним наслідком останнього твердження, очевидно, є

Твердження. Будь-яка повна система алгебри $A_D^{чр}$ містить хоча б одну некінечнозначну D-функцію.

Наприкінці відзначимо, що в ППА легко моделюються логічні зв'язки довільної складності через завжди істинний і завжди хибний предикати – p_T та p_F, відповідно, за допомогою часткого випадку операції $\Diamond_\beta^{m+1} - \Diamond$. Так, наприклад,

$$p(x_1,\ldots,x_n) \vee q(y_1,\ldots,y_m) = \Diamond\Big(p(x_1,\ldots,x_n), p_{\mathrm{T}}(x_1), \Diamond\big(q(y_1,\ldots,y_m), p_T(x_1), p_{\mathrm{F}}\big)\Big).$$

$$p(x_1,\ldots,x_n) \,\&\, q(y_1,\ldots,y_m) = \Diamond\Big(p(x_1,\ldots,x_n), \Diamond\big(q(y_1,\ldots,y_m), p_T(x_1), p_{\mathrm{F}}\big), p_{\mathrm{F}}(x_1)\Big).$$

$$\neg p(x_1,\ldots,x_n) = \Diamond\big(p(x_1,\ldots,x_n), p_{\mathrm{F}}, p_T(x_1)\big).$$

3.1.2 Повнота в класах чр-функцій та чр-предикатів. Критерій повноти

Розглянемо загальний метод знаходження повних систем ППА D-функцій і D-предикатів. Він буде представлений рядом взаємопов'язаних результатів, що представлені у вигляді доведених під час викладення лем та теорем. Спочатку визначимо деякі поняття та домовимось про ряд корисних позначень та домовленостей.

Зафіксуємо дві зліченні множини D_1 та D_2 за заданими для них ефективними нумераціями[9] $\alpha_1 : N \to D_1$, $\alpha_2 : N \to D_2$ і розглянемо ППА $A_{D_1}^{чр}$ та $A_{D_2}^{чр}$. Домовимось позначати елементи множин D_1 та D_2 маленькими літерами $a^1, b^1, \ldots$ и $a^2, b^2, \ldots$, можливо, з нижніми індексами відповідно. Нехай для алгебри $A_{D_1}^{чр}$ буде вирішена проблема повноти. Інакше кажучи, для неї буле задана її повна система σ_{D_1} та конструктивно задані ін'єктивні відображення $\phi : D_2 \to D_1$

[9] Ефективність нумерації $\alpha : N \to D$ означає, що конструктивно задана процедура відновлення а) за будь-яким номером $n \in N$ – відповідного нумерованого елементу $\alpha(n) \in D$ та б) та за будь-якомим елементом нумерованої множини $d \in D$ – його номеру $n \in N$ такого, що $\alpha(n) = d$.

та $\Phi: D_1 \to D_2$. Причому множини $\phi(D_2)$ та $\Phi(D_1)$ – рекурсивні [103] [67]. Розглянемо підхід до вирішення проблеми повноти для алгебри $A^{чр}_{D_2}$.

Для позначення D_1- та D_2-функцій використовуємо маленькі та прописні літери $f, g, ...$ та $F, G, ...$ відповідно, а для D_1- та D_2-предикатів – літери $p, r, ...$ та $P, R, ...$ відповідно. При використанні термальної форми запису змінні для D_1-функцій та D_1-предикатів будемо позначати маленькими літерами латинського алфавіту $x, y, z, ...$, а для D_2-функцій та D_2-предикатів – маленькими літерами грецького алфавіту $\tau, \xi, \pi, ...$ У всіх випадках можливе використання нижніх індексів.

Визначення. $\phi(D_2)$-функцію $f(x_1, ..., x_n)$ назвемо D_1-образом D_2-функції $F(\tau_1, ..., \tau_n)$, якщо $f(\phi(a_1^2), ..., \phi(a_n^2)) \cong \phi(F(a_1^2, ..., a_n^2))$ для будь-яких $a_1^2, ..., a_n^2 \in \phi(D_2), \phi(D_2) \subseteq D_1$.

Визначення. $\phi(D_2)$-предикат $p(x_1, ..., x_n)$ назвемо D_1-образом D_2-предиката $P(\tau_1, ..., \tau_n)$, якщо $p(\phi(a_1^2), ..., \phi(a_n^2)) \cong P(a_1^2, ..., a_n^2)$ для будь-яких $a_1^2, ..., a_n^2 \in \phi(D_2), \phi(D_2) \subseteq D_1$.

Покажемо, що задекларовані проведеними визначеннями відношення «бути образом функції» та «бути образом предикату» зберігають властивість часткової рекурсивності. Строго кажучи, справедлива

Теорема 3.1. D_1-образ D_2-чр-функції (D_2-чр-предикату) є D_1-чр-функцією (D_1-чр-предикатом).

Дійсно, зважаючи на ефективність нумерацій α_1 та α_2, а також конструктивність відображення ϕ легко перевірити, що ϕ як відображення нумерованої множини $\langle D_2, \alpha_2 \rangle$ на нумеровану множину $\langle \phi(D_2), \alpha_1(\alpha_2^{-1}(\phi(D_2))) \rangle$ є чр-еквівалентністю [67] [104]. Застосувавши теорему 2.1.5 [104], отримаємо, що справедлива

Лема. D_1-образом D_2-чр-функції (D_2-чр-предикату) є $\phi(D_2)$-чр-функція ($\phi(D_2)$-чр-предикат).

Звідси, а також із рекурсивності множини $\phi(D_2)$ безпосередньо слідує

Лема. Будь-яка $\phi(D_2)$-чр-функція є D_1-чр-функцією. Аналогічно для предикатів.

Звідси безпосередньо слідує справедливість теореми 3.1.

Визначення. D_2-чр-функцію $F(\tau_1,...,\tau_n)$ назвемо D_2-моделлю D_1-функції $f(x_1,...,x_n)$, якщо $F(\Phi(a_1^1),...,\Phi(a_n^1)) \cong \Phi(f(a_1^1,...,a_n^1))$ для будь-яких $a_1^1,...,a_n^1 \in D_1$. D_2-модель D_1-предиката вводиться аналогічним чином.

Нехай, $\psi = \phi \cdot \Phi$[10]. Очевидно що $\psi : D_2 \to \Phi(\phi(D_2))$ – бієкція. Тому коректно казати про обернене ψ відображення. Через χ позначимо деяке розширення відображення $\psi^{-1} : D_2 \to D_2$. Змістовно кажучи, D_2-функції ψ та χ грають ролі кодуючої та розкодуючої функцій відповідно. Позначимо через σ_{D_2} таку сукупність D_2-функцій та D_2-предикатів, що, по-перше, D_2-модель D_1-функції (D_1-предикату) з ПС σ_{D_1} може бути побудована з D_2-функцій та D_2-предикатів сукупності σ_{D_2} скінченним застосуванням операцій сигнатури Ω та, по-друге, D_2-функції ψ та χ аналогічним способом можуть бути побудовані з D_2-функцій та D_2-предикатів сукупності σ_{D_2}.

Визначення. Впорядковану шістку $\sum \equiv < D_1, D_2, \sigma_{D_1}, \sigma_{D_2}, \psi, \chi >$ назвемо допустимою системою (ДС), а пару $< D_1, \sigma_{D_1} >$ – її основою.

Очевидно, що в контексті введених кодуючої та розкодуючої функцій справедлива

Лема. Нехай $F(\tau_1,...,\tau_n)$ – D_2-чр-функція, а $H(\pi_1,...,\pi_n)$ – D_2-модель D_1-образу функції $F(\tau_1,...,\tau_n)$. Тоді $F(a_1^2,...,a_n^2) \cong \chi(H(\psi(a_1^2),...,\psi(a_n^2)))$ для будь-яких $a_1^2,...,a_n^2 \in D_2$.

---

[10] $f \cdot g$ є стандартним добутком функцій, інакше кажучи, така функція, що $dom(f \cdot g) \equiv dom(f)$ та $ran(f \cdot g) \subseteq ran(g)$, і яка будь-якому $d \in dom(f \cdot g)$ зіставляє значення $f \cdot g(d) \equiv g(f(d))$.

Лема. Нехай $P(\tau_1,...,\tau_n)$ – D_2 -чр-предикат, а $R(\pi_1,...,\pi_n)$ – D_2 -модель D_1 -образу предиката $P(\tau_1,...,\tau_n)$. Тоді $P(a_1^2,...,a_n^2) \cong R(\psi(a_1^2),...,\psi(a_n^2))$ для будь-яких $a_1^2,...,a_n^2 \in D_2$.

Звідси безпосередньо слідує, що справедлива

Теорема. σ_{D_2} – ПС алгебри $A_{D_2}^{чр}$.

Зважаючи на те, що елементи множин D_1 та D_2 – абстрактні об'єкти, в структуру яких ми не поглиблюємось, а до самих множин висуваються лише загальні вимоги, проведені побудови носять максимально загальний характер. Це дозволяє сформулювати просту, але ефективну умову повноти системи функцій ППА.

Отже, нехай $D_1, D_2, \sigma_{D_1}, \sigma_{D_2}, \psi, \chi$ – об'єкти, введені раніше. Тоді справедлива

Теорема. Якщо $\langle D_1, D_2, \sigma_{D_1}, \sigma_{D_2}, \psi, \chi \rangle$ є ДС, то σ_{D_2} – ПС алгебри $A_{D_2}^{чр}$.

Значимість цього результату полягає ще й у тому, що в абсолютно мотивованому припущенні справедливості тези Черча в його тлумаченні у контексті властивості нумераційної обчислюваності, теорема перетворюється в критерій повноти.

Теорема (критерій повноти). Щоб система σ_{D_2} була ПС алгебри $A_{D_2}^{чр}$, необхідно та достатньо, щоб система $\langle D_1, D_2, \sigma_{D_1}, \sigma_{D_2}, \psi, \chi \rangle$ була допустимою системою.

Отримані результати дають у своїй сукупності достатньо цілісне уявлення про метод побудови повних систем ППА частково-рекурсивних функцій і предикатів над зліченними множинами. Нижче цей метод буде застосований для вирішення проблеми повноти ППА в класі чр-функцій і чр-предикатів над прагматично значним у програмування типом даних – множиною записів.

3.1.3 ПА чр-функцій та чр-предикатів над множиною записів

Різноманітні інтуїтивні трактування запису дуже широко представлені в інформаційних технологіях і програмуванні. Не дивлячись на те, що окремі його розуміння достатньо сильно різняться, всім їм властиве намагання адекватно у тому чи іншому ступені використовувати для опису складно агрегованих сутностей іменні структури. Часто ці намагання обтяжені незначними частинними деталями, що розмивають у таких описах системоутворюючу значимість власне механізмів іменування. Хоча, як показує досвід, саме іменні структури тут складають той «спільний знаменник», через який варто розглядати всі інші аспекти вирішуваних задач. Саме це намагання лежить в основі наступних побудов.

Зафіксуємо V та W – непусті зліченні множини елементів, що тлумачяться відповідно як множини імен і значень (денотатів). У найширших розглядах допускається, що деякі імена можуть виступати в ролі значень і навпаки, інакше кажучи, можливо, що $V \cap W \neq \varnothing$.

Для подальшого викладення необхідно домовитись про деякі позначення, ввести основні та допоміжні визначення. Деякі з них дамо зараз, інші – будемо вводити за необхідністю. Всі невизначені поняття та позначення будуть трактуватись у сенсі [11].

Одним із основних понять цього розділу є запис. Множину всіх записів над множиною імен V та значень W позначимо $Z^{(V,W)}$. Тепер дамо визначення самого запису.

Визначення. Під записом над множиною імен V та значень W (просто записом, якщо з контексту ясно про які V та W йде мова) будемо розуміти скінченне функціональне бінарне відношення між множинами імен V та значень W.

Для позначення записів домовимось користуватись прописними літерами $I, J.K, \ldots$. Маленькими літерами $u, v, w, \ldots$ будемо позначати імена елементів записів, літерами $a, b, c, d, \ldots$– їх значення, а літерами $\lambda, \mu, \eta, \ldots$– власне елементи записів. У всіх випадках за необхідністю можливе використання індексів. У

випадку необхідності явно вказати в позначенні елементу запису його імені та (або) значення будемо використовувати відповідно верхній та (або) нижній ліві індекси. Наприклад, нехай $\lambda = (v, a)$. Тоді допустимі наступні варіанти позначення цього елементу: ${}^{v}\lambda$, ${}_{a}\lambda$ та ${}_{a}^{v}\lambda$.

У наступному викладенні корисним може бути використання наряду з записами їхніх так званих схем, які є іменними шаблонами відповідних записів.

Визначення. Під схемою запису K будемо розуміти скінченну множину імен $\{v_1,...,v_n\}$, яка являє собою проекцію цього запису за першою компонентністю, інакше кажучи, $\{v_1,...,v_n\} = pr_1(K)$, де pr_i функція проекції по i-й компоненті m-арного відношення $(1 \le i \le m)$ [11].

Домовимось схему запису I позначати $sh(I) = \{v_1,...,v_n\}$, а сам запис називати для компактності $sh(I)$ – записом або записом типу $sh(I)$. Записи, що мають однакові схеми домовимось називати односхемними записами. У разі необхідності явно вказати тип запису I будемо використовувати позначення $I^{sh(I)}$. Множину всіх записів типу $\{v_1,...,v_n\}$ позначимо $Z[\{v_1,...,v_n\}]$. Тоді матимуть місце такі частинні випадки: $I^{\varnothing} = \varnothing$ та $Z[\varnothing] = \{\varnothing\}$. Зважаючи на сказане, очевидно, що $Z^{(V,W)} \equiv \bigcup_{V' \in 2^V} Z[V']$. У розгорнутій формі будемо представляти запис як $I^{\{v_1,...,v_n\}} \equiv \{(v_1, d_1),...,(v_n, d_n)\}$.

Для коректного використання нумераційної обчислюваності на множині записів необхідно довести існування ефективної нумерації множини $Z^{(V,W)}$. Враховуючи зліченність множин V, W, а також те, що у цьому випадку не стільки важлива форма представлення імен і значень, а стільки їхні принципово різні ролі, можна не обмежуючи загальність наступних побудов вважати, що $V = W \equiv N$. Таким чино, усі подальші формальні побудови будемо проводити над множиною записів $Z^{(V,W)} \equiv Z^{(N,N)}$.

Здійснимо побудову нумерації в декілька кроків. По-перше, необхідно врахувати, що для будь-якого непустого запису $I = \{(v_1, d_1),...,(v_m, d_m)\}$ її номер

співпадає з номером скінченної множини $M_I = \{n_1,...,n_m\}$, де n_i є номером іменованого елементу запису (v_i, d_i)[11]. Для самої ж множини M_I її номер задається, наприклад, так: $\alpha'(M) \equiv 2^{n_{j_1}+1} \cdot 3^{n_{j_2}+1} \cdot ... \cdot p_{m-1}^{n_{j_m}+1}$, де $n_{j_1} < n_{j_2} < ... < n_{j_m}$, $n_{j_s} \in M, s = 1,...,m$, а p_i, $i = 0, 1, 2, ...$ – i-те просте число ($p_0 = 2$, $p_1 = 3$, $p_2 = 5$, …). Тоді шукана нумерація $\alpha_{Z^{(V,W)}}$ множини $Z^{(V,W)}$ задається через наступну кускову схему $\alpha'_{Z^{(V,W)}}(K) = \begin{cases} 1, якщо\ I = \varnothing \\ \alpha'(M_K), інакше \end{cases}$, де K – деякий запис. Через те, що $\alpha'_{Z^{(V,W)}}$, вочевидь, є бієкцією, можна вважати, що $\alpha_{Z^{(V,W)}} \equiv (\alpha'_{Z^{(V,W)}})^{-1}$.

Перейдемо безпосередньо до питання відшукання повної системи ППА чр-функцій і чр-предикатів над записами $A^{чр}_{Z^{(V,W)}}$. Зважаючи на отримані вище результати, вирішення проблеми повноти ППА $A^{чр}_{Z^{(V,W)}}$ зводиться до побудови відповідної допустимої системи. Для цього звернемось до поняття мультимножини, розглянутому, наприклад, у [105] [106]. Нехай U – деяка скінченна, можливо, пуста множина.

Визначення. Мультимножиною α з основою U будемо називати скінченновизначену функцію вигляду $\alpha : U \to N^+$, де N^+ – множина натуральних чисел без нуля, тобто $N^+ \equiv N \setminus \{0\} = \{1,2,3,...\}$. У разі необхідності вказати основу α, будемо використовувати для цієї мультимножини також позначення α^U.

Без обмеження загальності можна вважати, що $U \subseteq N$. Позначимо сукупність усіх мультимножин з основою U через M_U. Тоді, очевидно, що $M \equiv \bigcup_{U \in 2^N} M_U$ – множина усіх мультимножин (над N).

Домовимось елементи множини M позначати маленькими літерами грецького алфавіту $\alpha, \beta, \delta, ...$, можливо, з індексами. Елементи мультимножини

---

[11] Ефективна нумерація множини N^2 побудована, наприклад, в [].

будемо позначати парами вигляду $<a,d>$, кожен із компонентів може бути з індексом. Першу компоненту пари, a, будемо називати аргументом, другу – d – значенням (денотатом, кратністю). З мультимножинами зручно пов'язати поняття характеристики χ_α та повного образа $f[\alpha]$. Перша являє собою параметричну функцію $\chi_\alpha : D \to N$, значення якої задається кусковою схемою $\chi_\alpha(a) = \begin{cases} \alpha(a), \textit{якщо } a \in dom\ \alpha, \\ 0, \textit{інакше}; \end{cases}$ для всіх $a \in N$. Другий – по мультимножині α^U та відносно цієї функції $f : D \to D$ будує мультимножину $f[\alpha]^{f(U)}$, де $f(U)$ – повний образ множини U відносно функції f, а характеристика довільного аргументу a цієї множини задається наступним чином: $\chi_{f[\alpha]^{f(U)}}(a) = \sum_{a' \in f^{-1}(a)} \chi_\alpha(a')$. Тут $f^{-1}(a)$ – означає повний прообраз елемента a відносно функції f. У випадку пустої множини доданків сума вважається рівною нулю.

Далі перейдемо до розгляду ППА $A_M^{чр}$ M-чр-функцій і M-чр-предикатів. Цікавою є наступна сукупність M-чр-функцій та M-чр-предикатів σ_M, що включає *предикат рівності* $\alpha = \beta$, визначається, наприклад, так: $\alpha = \beta \Leftrightarrow \forall a(a \in N \Rightarrow \chi_\alpha(a) = \chi_\beta(a))$; *функцію об'єднання* $\bigcup_{All}$ таку, що мультимножинам α та β зіставляє таку множину $\alpha \bigcup_{All} \beta$, що підходить для будь-якого аргументу a, його характеристика рівна $\max(\chi_\alpha(a), \chi_\beta(a))$, тобто $\chi_{\alpha \cup_{All} \beta}(a) \equiv \max(\chi_\alpha(a), \chi_\beta(a))$; функцію *прямого з'єднання* $\otimes$, котра довільним мультимножинам α^{U_α} та β^{U_β} співставляє мультимножину $(\alpha^{U_\alpha} \otimes \beta^{U_\beta})^{U_\alpha \times U_\beta}$, характеристика аргументів $<a_1, a_2>$ якої задається так: $\chi(<a_1, a_2>) = \chi_{\alpha^{U_\alpha}}(a_1) \cdot \chi_{\beta^{U_\beta}}(a_2), \forall a_1, a_2 \in N$; функції *додавання* $\oplus$ та *віднімання* $\div$, що визначаються відповідно виразами $\alpha \oplus \beta = +[\alpha \otimes \beta]$ та $\alpha \div \beta = \dot{-}[\alpha \otimes \beta]$; константні функції $\{1^1\}(\alpha)$ та $\varnothing_m(\alpha)$, що фіксують відповідно мультимножини

$\{<1,1>\}$ та $\varnothing_m \equiv \alpha^{\varnothing}$; функцію *кратності* ϕ , що співставляє двом мультимножинам вигляду $\{<n,1>\}$ та $\{<r,1>\}$, мультимножину $\{<n,r>\}$; а також селекторні функції I_m^n. Значимість описаної вище сукупності σ_M полягає в тому, що справедлива

Теорема (про повноту мультимножинної ППА). Сукупність $\sigma_M \equiv \{=, \bigcup_{All}, \oplus, \div, \{1^1\}, \varnothing_m, \phi, I_m^n\}_{m=1,...,n}^{n=1,2.,3,...}$ є повною системою ППА $A_M^{чр}$ [105] [106].

Через подібність множини записів і мультимножин, нескладно побудувати ін'єктивне відображення множини записів у мультимножини $\phi : Z^{(V,W)} \to M$, котре визначається як $\phi(K) = \{<v_1, d_1 + 1>, <v_2, d_2 + 1>, ... <v_m, d_m + 1>\}$, де $m \in N$ – кількість елементів запису K. Обернене до нього відображення $\Phi : M \to Z^{(V,W)}$ будується аналогічним чином: $\Phi(\delta) = \{(a_1, d_1 - 1), (a_2, d_2 - 1), ...(a_m, d_m - 1)\}$, де δ – деяка мультимножина, а $m \in N$ – кількість елементів у ній. Крім того, очевидно, що множини $\phi(Z^{(V,W)})$ і $\Phi(M)$ рекурсивні.

Виходячи з вищесказаного, має місце

Лема. M -образ $Z^{(V,W)}$ -чр-функції ($Z^{(V,W)}$ -чр-предикату) є $\phi(Z^{(V,W)})$ -чр-функцією ($\phi(Z^{(V,W)})$-чр-предикатом).

З очевидної рекурсивності множини $\phi(Z^{(V,W)})$ можна зробити висновок, що будь-яка $\phi(Z^{(V,W)})$-чр-функція є M -чр-. Аналогічно для предикаті. Таким чином, справедливий

Наслідок. M -образ $Z^{(V,W)}$ -чр-функції ($Z^{(V,W)}$ -чр-предикату) є M -функцією (M -предикатом).

Розглянемо наступні функції та предикати на множині записів, навівши для окремих із них прості, але репрезентативні приклади застосування. Попередньо введемо корисну для подальших побудов допоміжну параметричну функцію проекції запису $pr_{\{v_{i_1},...,v_{i_k}\}}$, котра будь-якому запису, наприклад, $I = \{(v_1, d_1), ..., (v_n, d_n)\}$, ставить у відповідність новий запис

$pr_{\{v_{i_1},...,v_{i_k}\}}(I) \equiv \{(v_{j_1}, d_{j_1}),...,(v_{j_p}, d_{j_p})\}$, де $\{v_{j_1},...,v_{j_p}\} \equiv \{v_{i_1},...,v_{i_k}\} \cap \{v_1,...,v_n\}$ та $pr_{\{v_{i_1},...,v_{i_k}\}}(I) \subseteq I$. Отже, *предикат рівності* $=_Z$, вводиться аналогічно предикату рівності мультимножин; *видалення по зразку* $\div^Z$: $I \div^Z J \equiv pr_{pr_1(I)\setminus pr_1(J)}(I)$. Для окремого (на випадку $pr_1(I) \cap pr_1(J) = \varnothing$, $I \div^Z J = I$. Наприклад, $\{(1,3),(2,10),(5,7)\} \div^Z \{(1,1),(2,5),(3,7)\} = \{(5,7)\}$, $\{(5,7)\} \div^Z \{(6,5),(3,7)\} = \{(5,7)\}$, $\{(6,5),(3,7)\} \div^Z \varnothing = \{(6,5),(3,7)\}$ та $I \div^Z I = \varnothing, \ \forall I \in Z$; накладення записів ∇, що для будь-яких значень $I, J \in Z^{(V,W)}$ визначається як $I \nabla J \equiv J \bigcup pr_{pr_1(I)\setminus pr_1(J)}(I)$. У випадку, коли $I \nabla \varnothing = I, \varnothing \nabla J = \varnothing$, а у випадку $pr_1(I) \cap pr_1(J) = \varnothing$. Наприклад, $\{(1,3),(5,7)\} \nabla \{(1,1),(2,5),(3,7)\} = \{(1,1),(2,5),(3,7),(5,7)\}$; *включення в запис* $\overset{+}{U}$: $\overset{+}{U}(I) \equiv \begin{cases} I \cup \{\max(pr_1(I)) + 1, 0\}, \text{якщо } I \neq \varnothing \\ \{(0,0)\}, \text{якщо } I = \varnothing \end{cases}$, для всіх $I \in Z^{(V,W)}$. Наприклад, для $I = \{(1,3),(2,10)\}$ та $J = \varnothing$ отримаємо відповідно $\overset{+}{U}(I) = \{(1,3),(2,10),(3,0)\}$ та $\overset{+}{U}(J) = \{(0,0)\}$; *вибір по максимальному імені* max : $\max(I) \equiv pr_{\max(pr_1(I))}(I)$; обнулення значень $\{0\}$: $\{0\}(I) \equiv J$, де $pr_1(J) = pr_1(I) \,\&\, pr_2(J) = \{0\}$. Наприклад, $\{0\}(\ \{(1,3),(2,10)\}) = \{(1,0),(2,0)\}$; *інкремент* $\uparrow$: ставить у відповідність до будь-якого «непустого» запису $I \in Z^{(V,W)}$ запис $\uparrow(I)$ такий, що $\uparrow(I) = \{(v, a+1) \,|\, \forall (v,a) \in I\}$; *декремент* $\downarrow$: ставить у відповідність будь-якому «непустому» запису $I \in Z^{(V,W)}$ запис $\downarrow(I)$ такий, що $\downarrow(I) = \left\{(v,b) \,\middle|\, \forall (v,a) \in I \,\&\, b = \begin{cases} a-1, a > 0 \\ 0, a = 0 \end{cases}\right\}$. У випадку, якщо $I = \varnothing$, то і $\uparrow(I) = \downarrow(I) = \varnothing$.

Позначимо $\sigma_{Z^{(V,W)}} = \left\{=_Z, \nabla, \div^Z, \overset{+}{U}, \max, \{0\}, \uparrow, \downarrow, I_m^n\right\}, \begin{matrix} n=1,2,... \\ m=1,...,n \end{matrix}$ – множина $Z^{(V,W)}$ -чр-функцій та $Z^{(V,W)}$-чр-предикатів.

За аналогією до попереднього розділу розглянемо $Z^{(V,W)}$-функції ψ та χ – кодуючу та розкодуючу функції, відповідно, такі, що $\psi \equiv \phi \cdot \Phi$, а χ – деяке розширене відображення ψ^{-1}.

Мають місце наступні результати:

Лема. $Z^{(V,W)}$-модель M-функції (M-предикату), котра належить множині σ_M може бути побудована з функцій множини $\sigma_{Z^{(V,W)}}$ за допомогою операцій ППА.

Для M-предиката рівності та M-функцій $\bigcup_{All}, \div, \{1^1\}, \varnothing_m, \phi$ побудова їхніх $Z^{(V,W)}$-моделей не є складною. Тому не будемо зупинятись на них. Проведемо побудову для функції додавання мультимножин $\oplus$. Для цього введемо декілька допоміжних $Z^{(V,W)}$-функцій та $Z^{(V,W)}$-предикатів. А саме, *завжди хибний* та *завжди істинний* предикати: $Fal = S(=, I_1, S(\overset{+}{U}, I_1))$ та $Tru = S(=, I_1, I_1)$; *предикат нерівності* $Neq = \Diamond(S(=, I_1, I_2), Fal, Tru)$; *константа* $\varnothing^Z = S(\div^Z, I_1, I_1)$; *вибір за зразком Sel*: $Sel(I_1, I_2) = S^3(\div^Z, I_1, S^3(\div^Z, I_1, I_2))$. Наприклад, $Sel(\{(1,1),(2,2)\},\{(2,3),(4,5)\}) = \{(2,2)\}$ – функція «вибирає» з запису I_1, ті компоненти, імена яких присутні в якості імен компонентів у записі I_2. Отже, I_2 є таким собі зразком для вибірки з I_1; максимальне додавання односхемних записів: $+^{\max} = *^3 S^3(Neq(I_2^2, S^2(\{0\}, I_2^2)), S^2(\uparrow, I_1^2), S^2(\downarrow, I_2^2))$. Так, наприклад, для записів $I_1 = \{(1,1),(2,2)\}$ та $I_2 = \{(1,3),(2,5)\}$ отримаємо $+^{\max}(I_1, I_2) = \{(1,1+5),(2,2+5)\} = \{(1,6),(2,7)\}$. Слід зауважити, що операція максимального додавання загалом не комутативна $+^{\max}(I_1, I_2) \neq +^{\max}(I_2, I_1)$. Комутативність зберігається тільки для записів спеціального вигляду, наприклад, односхемних одноелементних записів. Наприклад, якщо $I_1 = \{(2,2)\}$ та $I_2 = \{(2,5)\}$, то $+^{\max}(I_1, I_2) = +^{\max}(I_2, I_1) = \{(2,7)\}$.

Тепер перейдемо безпосередньо до побудови $Z^{(V,W)}$-функції $\oplus^Z$-моделі M-функції $\oplus$. Нехай ці записи $I_1 = \{(v_1^1, d_1^1), ..., (v_{n_1}^1, d_{n_1}^1)\}$ та $I_2 = \{(v_1^2, d_1^2), ..., (v_{n_2}^2, d_{n_2}^2)\}$

такі, що. Тоді схематично операція $\oplus^Z$ може бути продемонстрована (таблиця 3.1)

Таблиця 3.1 Схема операції $\oplus^Z$.

		v_1^1	v_2^1	v_3^1	$\cdots$	v_{r_1}	$\cdots$	v_{r_s}	$\cdots$	$v_{n_1-2}^1$	$v_{n_1-1}^1$	$v_{n_1}^1$					
		d_1^1	d_2^1	d_3^1	$\cdots$	$d_{r_1}^1$	$\cdots$	$d_{r_s}^1$	$\cdots$	$d_{n_1-2}^1$	$d_{n_1-1}^1$	$d_{n_1}^1$					
		I_{1_1}				I_{1_2}				I_{1_1}							
v_1^2	v_2^2	$\cdots$				v_{r_1}	$\cdots$	v_{r_s}	$\cdots$						$v_{n_2-1}^2$	$v_{n_2}^2$	
d_1^2	d_2^2	$\cdots$				$d_{r_1}^2$	$\cdots$	$d_{r_s}^2$	$\cdots$						$d_{n_2-1}^2$	$d_{n_2}^2$	
I_{2_1}						I_{2_2}									I_{2_1}		
v_1^2	v_2^2	v_1^1	v_2^1	v_3^1	$\cdots$	v_{r_1}	$\cdots$	v_{r_s}	$\cdots$	$v_{n_1-2}^1$	$v_{n_1-1}^1$	$v_{n_1}^1$			$v_{n_2-1}^2$	$v_{n_2}^2$	
d_1^2	d_2^2	d_1^1	d_2^1	d_3^1	$\cdots$	$d_{r_1}^1 + d_{r_1}^2$	$\cdots$	$d_{r_s}^1 + d_{r_s}^2$	$\cdots$	$d_{n_1-2}^1$	$d_{n_1-1}^1$	$d_{n_1}^1$			$d_{n_2-1}^2$	$d_{n_2}^2$	

Очевидно, що ця операція «розбиває» записи I_1 та I_2 на «ділянки», позначені I_{1_1}, I_{1_2}, I_{2_1} та I_{2_2} зі схемами $sh(I_{1_1}) = sh(I_1) \setminus \{v_{r_1},...,v_{r_s}\}$, $sh(I_{1_2}) = \{v_{r_1},...,v_{r_s}\}$ та $sh(I_{2_2}) = \{v_{r_1},...,v_{r_s}\}$. Таким чином, результуючий запис I_1 може бути представлений як $I_3 = I_{1_1} \cup I_{2_1} \cup I_{1_2} \oplus^Z I_{2_2}$. Щодо перших двох «доданків» I_{1_1} та I_{2_1}, вони легко отримуються за допомогою раніше введеної функції $\div^Z$. А саме, $I_{1_1} = I_1 \div I_2$ та $I_{2_1} = I_2 \div I_1$. Що ж до $I_{1_2} \oplus^Z I_{2_2}$, то самі I_{1_2} та I_{2_2} легко задаються за допомогою функції Sel: $I_{1_2} = Sel(I_1, I_2)$ та $I_{2_2} = Sel(I_2, I_1)$. Залишається промоделювати $Z^{(V,W)}$-функцію $\oplus^Z$ для односхемних записів. Неважко переконатись, що $\oplus^Z$ може бути представлена, наприклад, так:

$$\oplus^Z = *^4(\underbrace{Neq(I_1^2, I_2^2)}_{p}, \underbrace{I_1^2 \nabla(\max(I_2^2) +^{\max} (Sel(I_1^2, \max(I_2^2)))}_{f_1}, \underbrace{I_2^2 \div^Z \max(I_2^2)}_{f_2}).$$

Очевидно, що у випадку, коли $sh(I_1) \cap sh(I_2) = \varnothing$, запис $I_{1_2} \oplus^Z I_{2_2}$ теж є порожнім, отже, $I_{1_2} \oplus^Z I_{2_2} = \varnothing$ та, як наслідок, результат $I_3 \equiv I_1 \oplus^Z I_2 = I_{1_1} \cup I_{2_1}$. Враховуючи, що $sh(I_{1_1}) \cap sh(I_{2_1}) = \varnothing$ по побудові, очевидно, що $I_3 \equiv I_1 \oplus^Z I_2 = I_1 \nabla I_2 = I_{1_1} \nabla I_{2_1}$.

Лема. Функції ψ та χ можуть бути побудовані з функцій сукупності $\sigma_{Z^{(V,W)}}$ скінченним застосуванням операцій ППА.

Справедливість цього результату очевидна внаслідок згаданої подібності записів та мультимножин, простоти кодуючого та розкодуючого відображень ϕ та Φ, а також проведених вище побудов. Тому справедлива

Лема. $< M, Z^{(V,W)}, \sigma_M, \sigma_{Z^{(V,W)}}, \psi, \chi >$ – допустима система.

Звідси та зважаючи на вищезгадане безпосередньо випливає справедливість теореми 3.2.

Теорема 3.2. $\sigma_{Z^{(V,W)}} = \left\{ =_Z, \nabla, \div^Z, \overset{+}{U}, \max, \{0\}, \uparrow, \downarrow, I_m^n \right\}, \begin{matrix} n=1,2,... \\ m=1,...,n \end{matrix}$ – породжуючи система ППА $A^{чр}_{Z^{(V,W)}}$.

3.2 ППА в класі функцій над записами, що зберігають денотати

3.2.1 Загальні положення, визначення та позначення

Тут будуть змістовно розглянуті та строго визначені поняття вектору та запису, а також введені окремі важливі для подальших побудов функції над ними. Далі в межах ППА буде наведена алгебраїчна характеристика класу частково-рекурсивних маніпуляційних функцій і частково-рекурсивних предикатів. Побудови будуть спиратись на отриману в [107] алгебраїчну характеристику класу обчислюваних маніпуляційних функцій та обчислюваних предикатів над векторами, а також на результати роботи [7].

Вектори, записи та функції над ними. Не обмежуючи загальність побудов, що випливає, наприклад, з результату по нумераційній обчислюваності [104] та результатів дослідження ППА арифметичних функцій [90], домовимось далі розуміти під векторами вектори натуральних чисел, а під записами – записи натуральних чисел, які позначатимуться відповідно N^* та $Z^{(N,N)}$ [7] [107]. Тоді $N^* = \bigcup_{i=0}^{\infty} N^i$, де $N^0 = \varnothing$ – так званий нульовий вектор, $N^1 = N$, а $N = \underbrace{N \times ... \times N}_{i}, i \in 2,3,...$. Для позначення векторів будемо використовувати прописні літери $P, Q, R, ...$, а елементи векторів будемо позначати рядковими літерами $p, q, r...$. При цьому допускається використання індексів у позначеннях. Для позначення конкретного вектору будемо використовувати (залежно від ситуації) записи вигляду $< p_1, p_2, ..., p_n >$ та $\overline{p_1 p_2 ... p_n}$, розуміючи їх еквівалентними, інакше кажучи, $< p_1, p_2, ..., p_n > \equiv \overline{p_1 p_2 ... p_n}$. Пустий вектор (вектор, що не містить елементів) будемо позначати Λ . У якості оцінки векторного універсуму зафіксуємо функцію $\beta : N^* \to 2^N_{fin}$, де 2^N_{fin} – множина усіх скінченних підмножин множини N і таку, що: $\beta(\Lambda) = \Lambda$, $\beta(< p_1, p_2, ..., p_n >) = \{p_1, p_2, ..., p_n\}$.

Отже, під *записом* над множинами імен та значень із N (далі просто записом) будемо розуміти скінченне функціональне бінарне відношення на множині N . Для позначення записів домовимось користуватися прописними літерами $I, J.K, ...$ Рядковими літерами $u, v, w, ...$ будемо позначати імена елементів записів, літерами $a, b, c, d, ...$ – їх значення, а літерами $\lambda, \mu, \eta, ...$ – власне елементи записів – пари, які складаються з імені та значення. В усіх випадках у разі потреби можна використовувати індекси. Оцінка β на записах задається аналогічно випадку з векторами: $\beta : Z^{(N,N)} \to 2^N_{fin}$, де 2^N_{fin} має такий же сенс, що і раніше згадувалось, а $\beta(\{\varnothing\}) = \varnothing$.

Зазначимо, що будь-який вектор можна промоделювати за допомогою деякого спеціального запису. Можливе також обернене представлення запису деяким спеціального вигляду вектором. Такі представлення будуть досить важливими у наступних побудовах. Тому їх варто розглянути окремо.

Нехай $< p_1, p_2,..., p_n >$ – довільний вектор. Тоді запис, що його представляє (моделює), може виглядати, наприклад, так: $\{(1, p_1),...,(n, p_n)\}$. Інакше кажучи, вектор легко моделюється записом із «стандартними» іменами від 1 до n. Ім'я тут не просто іменує, але й фіксує «стандартну» позицію кожного елемента вектору. Саме у зв'язку з цим таке іменування ми будемо називати *стандартним*, а імена, відповідно – «стандартними», інакше кажучи, такими, що відповідають стандарту вектору.

Крім того, вектор, що моделює довільний запис $\{(\nu_1, d_1),...,(\nu_n, d_n)\}$ буде виглядати як $< 0^{\nu_1+1}1^{d_1+1}.....0^{\nu_n+1}1^{d_n+1} >$, де p^k позначає k-мірний вектор $\underbrace{p...p}_{k}$. Наприклад, запис $\{(4,3),(1,2)\}$ буде представлений вектором $\overline{\underbrace{00000}_{4+1}\underbrace{111}_{3+1}\underbrace{0}_{1+1}\underbrace{0}_{1+1}\underbrace{11}_{2+1}}$, а $\{(1,2),(2,3)\}$ – вектором $\overline{0011100011\ 11}$.

Тепер введемо корисні функції на векторах і записах. Спочатку звернемось до векторів. Нехай $\sigma_{N^*} = \{\bar{p}\}_{p=1,2,...} \cup \{\text{H}, \text{E}, \Pi, \text{o}, =_{N^*}, I_m^n\}_{m=1,...n}^{n=1,2,...}$ – множина чр-функцій та чр-предикатів на векторах таких, що $H(\bar{n}) = \overline{\underbrace{0...0}_{n+1}} \equiv 0^{n+1}$, $H(\Lambda) = \Lambda$, $n \in N$ – функція відміток; функція $\text{E}(\Pi)$ – вибір (видалення) першої компоненти вектор, o– операція конкатенації двох векторів, а $=_{N^*}$ – предикат рівності двох векторів [107].

Звернемося до записів. Зауважимо, що через те, що множина імен записів є частково впорядкованою, не буде зайвим вважати, що елементи будь-якого непустого запису впорядковані за іменами. Розглянемо наступну сукупність

$Z^{(N,N)}$ -чр-функцій і $Z^{(N,N)}$ -чр-предикатів на записах: $\sigma_{Z^{(N,N)},\beta} = \{=_Z, Od, Od^{-1}, Zn, \nabla, \div^Z, Mn, Rn, Nm^+, \{(1,0)\}, I_m^n\}_{m=1,..n}^{n=1,2,...}$.

Функція стандартизації імен Rn перетворює будь-який запис у запис зі стандартними іменами відповідно до початкової впорядкованості елементів вихідного запису:

$Rn(\{(\nu_1, d_1),...,(\nu_n, d_n)\}) = \{(1, d_1),(2, d_2),...,(n, d_n)\}\big|_{\nu_1<\nu_2<...<\nu_n}$, $Rn(\{(\nu, d)\}) = \{(1, d)\}$, $Rn(\varnothing) = \varnothing$.

Функція кодування денотату Od ставить у відповідність будь-якому непорожньому запису новий запис, котрий є «одиничним кодом» елемента запису з найменшим іменем. Порожній запис перетворюється сам у себе, інакше кажучи, $Od(\{(\nu_1, d_1),...,(\nu_n, d_n)\})\big|_{\nu_1<\nu_2<...<\nu_n} = \{(1,1),(2,1),...,(d_1+1,1)\}$, $Od(\varnothing) = \varnothing$. Пояснимо сказане на прикладах: $Od(\{(2,4)\}) = \{(1,1),(2,1),(3,1),(4,1),(5,1)\}$ – модель вектору-коду $<1,1,1,1,1>$; $Od(\{(3,2),(4,6),(5,1)\}) = \{(1,1),(2,1),(3,1)\}$ – модель вектору-коду $<1,1,1>$; $Od(\varnothing) = \varnothing$.

Функція декодування денотату Od^{-1}. Змістовно ця функція є оберненою до тільки-но приведеної Od . Формально:

$Od^{-1}(L, \{\underbrace{(\nu_1,1),(\nu_2,1),...,(\nu_k,1)}_{\text{те, що декодується}}, (\nu_{k+1}, d_1),...,(\nu_n, d_n)\}) = \underbrace{\{(\nu_1, k-1)}_{\text{результат декодування}}, (\nu_{k+1}, d_1),...,(\nu_n, d_n)\}$, де L – деякий запис, котрий містить денотат $k-1$, $\nu_1 < \nu_2 < ... < \nu_k < ... < \nu_n$, $d_1 \in N \setminus \{1\}$. Наприклад,

$Od^{-1}(\{(1,2),(2,3)\}, \{\underbrace{(1,1),(2,1),(3,1)}_{\text{те, що декодується}}, (4,0),(5,0),(6,1)\}) = \{(1,2),(4,0),(5,0),(6,1)\}$;

$Od^{-1}(\{(3,5)\}, \{\underbrace{(1,1),(2,1),(3,1),(4,1),(5,1),(6,1)}_{\text{те, що декодується}}\}) = \{(1,5)\}$.

Функція кодування імені Zn. Змістовно, ця функція аналогічна введеній вище функції Od , але кодується вже ім'я, а не денотат. Строго кажучи:

$Zn(\{(\nu_1, d_1),...,(\nu_n, d_n)\})\big|_{\nu_1<\nu_2<...<\nu_n} = \{(1,0),(2,0),...,(\nu_1+1,0)\}$, $Zn(\varnothing) = \varnothing$.

Наприклад: $Zn(\{(3,2),(4,6),(5,1)\}) = \{(1,0),(2,0),(3,0),(4,0)\}$.

Функція видалення за зразком $\div^Z$: $\div^Z(I,J) = pr_{pr_1(I)\setminus pr_1(J)}(I)$. Інакше кажучи, ця функція видаляє з першого аргументу всі елементи, імена яких містяться у другому аргументі – зразку. Для частинного випадку $pr_1(I) \cap pr_1(J) = \varnothing$, $\div^Z(I,J) = I$. Тут pr_i – функція проекції по i-й компоненті $(1 \le i \le m)$ m-арного відношення [107], а $pr_{\{\nu_{i_1},...,\nu_{i_k}\}}$ – допоміжна параметрична функція проекції запису $pr_{\{\nu_{i_1},...,\nu_{i_k}\}}$, котра будь-якому запису, наприклад, $I = \{(\nu_1, d_1),...,(\nu_n, d_n)\}$ ставить у відповідність новий запис $pr_{\{\nu_{i_1},...,\nu_{i_k}\}}(I) \equiv \{(\nu_{j_1}, d_{j_1}),...,(\nu_{j_p}, d_{j_p})\}$, де $\{\nu_{j_1},...,\nu_{j_p}\} \equiv \{\nu_{i_1},...,\nu_{i_k}\} \cap \{\nu_1,...,\nu_n\}$ та $pr_{\{\nu_{i_1},...,\nu_{i_k}\}}(I) \subseteq I$. Продемонструємо роботу $\div^Z$ на прикладах:

$\div^Z(\{(1,3),(2,10),(5,7)\},\{(1,1),(2,5),(3,7)\}) = \{(5,7)\}$;

$\div^Z(\{(5,7)\},\{(6,5),(3,7)\}) = \{(5,7)\}$; $\div^Z(\{(6,5),(3,7)\},\varnothing) = \{(6,5),(3,7)\}$;

$\div^Z(I,I) = \varnothing$.

Вибір елемента з мінімальним іменем Mn*:*

$Mn(\{(\nu_1, d_1),...,(\nu_n, d_n)\})\big|_{\nu_1<\nu_2<...<\nu_n} = \{(\nu_1, d_1)\}$. У випадку з пустим записом, функція повертає пустий запис $\varnothing$: $Mn(\varnothing) = \varnothing$.

Функція слідування імен Nm^+. Збільшує імена елементів непустого запису на одиницю: $Nm^+(\{(\nu_1, d_1),...,(\nu_n, d_n)\}) = \{(\nu_1+1, d_1),(\nu_2+1, d_2),...,(\nu_n+1, d_n)\}$. Для пустого запису: $Nm^+(\varnothing) = \varnothing$.

Функція накладання ∇ . Для будь-яких $I, J \in Z^{(N,N)}$ $\nabla(I,J) = J \cup pr_{pr_1(I)\setminus pr_1(J)}(I)$. У часткових випадках: $\nabla(I,\varnothing) = I$, $\nabla(\varnothing,J) = \varnothing$, а у випадку, коли $pr_1(I) \cap pr_1(J) = \varnothing$, $\nabla(I,J) = \nabla(J,I) = J \cup I$. Наприклад, $\nabla(\{(1,3),(5,7)\},\{(1,1),(2,5),(3,7)\}) = \{(1,1),(2,5),(3,7),(5,7)\}$.

Функція ініціалізації $\{(1,0)\}$ ставить у відповідність будь-якому аргументу запис $\{(1,0)\}$.

Предикат рівності $=_Z$ вводиться як звуження на множину $Z^{(N,N)}$ стандартного предиката рівності множин.

3.2.2 Повнота ППА у класі чр-функцій, що зберігають денотати та чр-предикатів над записами

Повнота ППА у класі чр-функцій, що зберігають денотати та чр-предикатів над множиною записів буде доведена з використанням результату про повноту ППА у класі чр-функцій, що зберігають денотати, та чр-предикатів над множиною векторів [107].

ППА чр-функцій, що зберігають денотати, та чр-предикатів над записами та векторами позначимо відповідно як $A^{чр}_{Z^{(V,W)},\beta}$ та $A^{чр}_{N^*,\beta}$. Також згадаємо, що [107] $\sigma_{N^*,\beta} = \{\overline{p}\}_{p=1,2,\ldots} \cup \{H, E, \Pi, \circ, =_{N^*}, I^n_m\}^{n=1,2,\ldots}_{m=1,\ldots n}$ – система утворюючих ППА $A^{чр}_{N^*,\beta}$. Нехай $\varphi : Z^{(N,N)} \to N^*$ та $\Phi : N^* \to Z^{(N,N)}$ наступні ін'єкції:

$$\varphi(\{\varnothing\}) = (\varnothing),\ \varphi(\{(\nu_1, d_1),\ldots,(\nu_n, d_n)\}) = \begin{cases} (d_1 d_2 \ldots d_n), \textit{якщо } \nu_i = i, i = \overline{1,n} \\ (0^{\nu_1+1} 1^{d_1+1} \ldots .. 0^{\nu_n+1} 1^{d_n+1}), \textit{інкаше} \end{cases};$$

$$\Phi((\varnothing)) = \{\varnothing\},\ \Phi((p_1,\ldots,p_n)) = \{(1,p_1),\ldots,(n,p_n)\}.$$

Змістовно кажучи, ці функції використовують очевидне співвідношення вектору, наприклад, $(d_1 d_2 \ldots d_n)$, та відповідного запису зі стандартними іменами у цьому випадку запису $\{(1,d_1),(2,d_2),\ldots(n,d_n)\}$, і навпаки. Таким чином, якщо аргументом φ є запис зі стандартними іменами, то він відображається у вектор, який моделює. Інакше будується вектор, що його кодує. за вказаним вище правилом. Функція Ф будує модель вектору $< d_1, d_2, \ldots, d_n >$ у множині $Z^{(N,N)} - \{(1,d_1),(2,d_2),\ldots(n,d_n)\}$. Нижче ці функції грають роль відображень, одне з яких кодує, а інше – розкодує.

Означення 1. N^*-чр-функцію $f(P_1,P_2,...P_n)$, $n\in N$ називають N^*- образом $Z^{(N,N)}$- чр-функції $F(I_1,I_2,...I_n)$, якщо $f(\varphi(I_1),\varphi(I_2),...,\varphi(I_n))\cong\varphi(F(I_1,I_2,...,I_n))$ для всіх $I\in Z^{(N,N)}$. Аналогічно для предикатів.

Означення 2. $Z^{(N,N)}$-чр-функцію $F(I_1,I_2,...I_n)$, $n\in N$ назвемо $Z^{(N,N)}$-моделлю N^* -чр-функції $f(P_1,P_2,...,P_n)$, якщо $F(\Phi(P_1),\Phi(P_2),...\Phi(P_n))\cong\Phi(f(P_1,P_2,...,P_n))$ для будь-яких $P_1,P_2,...,P_n\in N^*$. $Z^{(N,N)}$-модель N^*-чр-предиката вводиться аналогічним чином.

Виходячи з визначень N^*-образа $Z^{(N,N)}$-чр-функції, $Z^{(N,N)}$-моделі N^*-чр-функції, повноти сукупності $\sigma_{N^*,\beta}$, можна встановити, що справедливі

Лема. Для будь-якої $Z^{(N,N)}$ -чр-функції існує її N^* -образ, який є N^* -чр-функцією. Те ж саме для предикатів.

Лема. Для будь-яких векторних чр-функцій і чр-предикатів існують їх $Z^{(N,N)}$-моделі, які належать до замикання $[\sigma_{Z^{(N,N)},\beta}]_\Omega$, де $\Omega-$ множина композицій ППА.

Доведення проводяться індукцією за довжиною відповідних константних термів N^* та $Z^{(N,N)}$ ППА.

Позначимо $\Psi: Z^{(N,N)}\to\Phi(\varphi(Z^{(N,N)}))-$ наступну бієкцію:

$$\Psi(\{(v_1,d_1),(v_2,d_2)...(v_n,d_n)=\begin{cases}\{(v_1,d_1),(v_2,d_2)...(v_n,d_n)\},\ якщо\ v_i=i,i=\overline{1,n}\\ \{\underbrace{(1,0),...,(k,0)}_{v_1+1},\underbrace{(k+1,1),...,(l,1)}_{d_1+1},\underbrace{(l+1,0),...,(r,0)}_{v_2+1},\\ \underbrace{(r+1,1),...,(s,1)}_{d_2+1},(s+1,0),...,\\ \underbrace{(m,0),...,(p,0)}_{v_n+1},\underbrace{(p+1,1),...,(o,1)}_{d_n+1}\},\ інакше\end{cases}$$, де

$(k<l<r<s<m<p<o)\in N$. Зауважимо, що, якщо аргументом є модель вектору – запис зі стандартними іменами, значення функції дорівнює самому

значенню аргументу. Таким чином, функція Ψ зберігає денотати та має характеристику $\{0,1\}$.

Нехай χ – деяке розширення $Z^{(N,N)}$ -чр-функції Ψ^{-1} . Додатковим ускладненням на шляху створення такої функції є збереження денотатів. Адже така функція може мати нескінченну характеристику – аргументом функції буде запис із денотатами з множини $\{0,1\}$, а результатом функції може бути запис із будь-якими денотатами. Бля вирішення цієї ситуації вводиться додатковий аргумент, який містить необхідні денотати. Таким чином, функція χ може бути визначена наступним чином:

$$K = \chi(I,\{(v_1,d_1),...,(v_n,d_n)\}) = \begin{cases} \Psi^{-1}, \textit{якщо } \beta(K) \in \beta(I) \\ \{(v_1,d_1),...,(v_n,d_n)\}, \textit{інакше} \end{cases}$$

Лема. Нехай $F(I_1,I_2,...,I_n)$ – $Z^{(N,N)}$ -чр-функція, яка зберігає денотати та $\{d_1,d_2,...,d_k\}, k \geq 0$ – її характеристика. Нехай $G(J_1,J_2,...,J_n)$ – $Z^{(N,N)}$ -модель векторного образа функції $F(I_1,I_2,...,I_n)$. Тоді

$$F(K_1,K_2,...,K_n) \cong \chi(L,G(\Psi(K_1),\Psi(K_2),...,\Psi(K_n)))$$

для всіх $K_1,K_2,...,K_n \in Z^{(N,N)}$. Аналогічно для предикатів.

Отже, для отримання результату по повноті ППА $A^{чр}_{Z^{(N,N)},\beta}$ залишається показати, що справедлива наступна лема

Лема 3.1 $\Psi,\chi,\Pi_Z,\mathrm{E}_Z,\mathrm{H}_Z,\mathrm{o}_Z \in [\sigma_{Z^{(N,N)},\beta}]_\Omega$, де $\Pi_Z,\mathrm{E}_Z,\mathrm{H}_Z\ \mathrm{o}_Z - Z^{(N,N)}$ - образи N^* -чр-функцій $\Pi,\mathrm{E},\mathrm{H},\mathrm{o}$.

Визначимо образи, які необхідно промоделювати. Функції H та E вже мають свої образи в $\sigma_{Z^{(N,N)},\beta}$, а саме - Zn та Mn. Образ Π_Z можна задати як результат віднімання двох множин: $\Pi_Z(\lambda) = \lambda \setminus \mathrm{E}_Z(\lambda)$. Образ функції конкатенації можна визначити так:

$$\mathrm{o}_Z\,(\{(\nu_1,d_1),...,(\nu_n,d_n)\},\{(u_1,c_1),...,(u_m,c_m)\})\Big|_{\nu_1<\nu_2<..<\nu_n\ \ u_1<u_2<..<u_m} =$$
$$=\{(\nu_1,d_1),...,(\nu_n,d_n)\}\cup\{(u_1+\nu_n,c_1),...,(u_m+\nu_n,c_m)\}.$$

Для доведення леми 3.1 необхідно промоделювати за допомогою функцій $\sigma_{Z^{(N,N)},\beta}$ $Z^{(N,N)}$-образи N^*-функцій з $\sigma_{N^*,\beta}$. Проведемо таке моделювання для кожної з вказаних функцій.

Функція Π_Z: $\Pi_Z = S^2(Rn, S^3(\div, Mn, I_1^1))$.

Конкатенація o_Z. Спочатку введемо декілька допоміжних функцій: отримання пустої множини $\varnothing_Z = S^3(\div, I_1^1, I_1^1)$, завжди хибний предикат $False = S^3(=_Z, \varnothing_Z, \{(1,0)\})$, завжди істинний предикат $True = S^3(=_z, \{(1,0)\}, \{(1,0)\})$, предикат нерівності $Neq = \Diamond(=_Z, False, True)$. Тоді сама функція конкатенації буде визначена як $\mathrm{o}_Z = S^3(\nabla, I_1^2, S(*^3(S^3(Neq, \varnothing_Z, S^3(\div, I_2^2, I_1^2)), Nm^+, I_2^2), I_2^2, I_2^2))$.

Функцію Ψ можна представити композицією з предиката та двох функцій: $\Psi = \Diamond(StNames, I_1^1, notStNames)$, де *notStNames* – предикат, що перевіряє, чи є аргумент моделлю вектору, *notStNames* – функція, що будує модель вектору, який кодує запис-аргумент. У випадку, коли аргумент функції Ψ – модель вектору, він і є результатом функції, інакше результатом функції є значення *notStNames* на її аргументі.

Предикат *StNames* визначається як

$$StNames = S^5(*^5(S^3(=_Z, \varnothing_Z, S^2(\Pi_Z, I_2^4)), \Diamond(S^3(=_Z, I_1^4, False)), False,$$
$$\Diamond(S^3(=_Z, S^2(E_Z, I_2^4), I_3^4), True, S^3(Neq, S^2(\mathrm{E}_Z, I_2^4), I_4^4)),$$
$$S^2(\Pi_Z, I_2^4), Nm^+, Nm^+), True, I_1^1, \{(1,0)\}, S^2(Od, \{(1,0)\})).$$

Для побудови функції *notStnames* знадобиться функція *convert*. Очевидно, що вона може бути представлена таким термом: $convert = S^3(\mathrm{o}_z, S^2(Zn, I_2^2), S^2(Od, I_2^2))$. Тоді: $notStnames = S^3(*^3(S^3(Neq, \varnothing_Z, I_2^2), S^3(\mathrm{o}_z, I_1^2, convert), S^2(\Pi_Z, I_2^2)), \varnothing_Z, I_1^2)$.

Функція χ є композицією двох функцій і предиката: $\chi = \Diamond(isModel, restore, I_1^1)$. Предикат $isModel$, що перевіряє, чи аргумент є моделлю коду запису, якщо його значення True, за цією моделлю будується вихідний запис, у випадку хиби повертається сам аргумент.

Предикат $isModel$ істинний за умови одночасного виконання наступних умов: денотат першого елемента запису дорівнює нулю ($FirstZero$), денотат останнього елемента запису дорівнює одиниці ($LastOne$), запис має стандартні імена ($StNames$), у записі всі денотати мають значення 1 або 0 ($Denotates10$), імена елементів запису, закодовані вектором-кодом, котрий моделюється записом-аргументом, впорядковані за зростанням ($AscNames$). Неважко переконатись, що ці умови можуть бути представлені у вигляді наступних $Z^{(N,N)}$ -чр-предикатів:

$$FirstZero = S^3(=_Z, \{(1,0)\}, E_Z),$$

$$LastOne = S^3(*^3(S^3(=_Z, \varnothing_Z, S^2(\Pi_Z, I_2^2)),$$
$$S^3(=_Z, S^2(Od, \{(1,0)\}), S^2(Rn, I_2^2)), S^2(\Pi_Z, I_2^2)), False, I_1^1),$$

Визначимо предикат $Denotates10$. Передусім задамо два допоміжні предикати, котрі визначають приналежність денотата, що розглядається, до множини $\{0,1\}$. Це а) предикат рівності денотата одиниці $Eq1 = S^3(=_Z, S^2(Od, \{(1,0)\}), S^2(\mathrm{E}_Z, S^2(Rn, I_2^2)))$ та b) предикат рівності його нулю $Eq0 = S^3(=_Z, \{(1,0)\}, S^2(\mathrm{E}_Z, S^2(Rn, I_2^2)))$. Тоді власне предикат $Denotates10$ визначатиметься так:

$$Denotates10 = S^3(*^3(S^3(Neq, \varnothing_Z, S^2(\Pi_Z, I_2^2)), \Diamond(S^3(=_Z, I_1^2, False), False,$$
$$S^3(=_Z, I_1^2, \Diamond(Eq1, True, Eq0))), S^2(\Pi_Z, I_2^2)), True, I_1^1).$$

Також визначимо предикат $AscNames$:

$$AscNames = \Diamond(S^3(=_Z, \varnothing_Z, I_1^1), False, S^3(*^3(\Diamond(S^3(Neq, S^2(NextPair, I_2^2), \varnothing_Z),$$
$$S^3(GtName, S^2(NextPair, I_2^2), I_2^2), False), S^3(GtName,$$
$$S^2(NextPair, I_2^2), I_2^2), S^2(NextPair, I_2^2)), True, I_1^1)).$$

Тут функція *NextPair* видаляє «ділянку» моделі коду запису, яка є представленням елементу закодованого запису та розташована «на початку» моделі коду, вона є композицією двох функцій: $NextPair = S^2(Rn, S^2(delO, delZ))$, де *delZ* видаляє «код» імені елемента, що видаляється, *delO*– «код» його денотату, при цьому виконується стандартизація імен в частині коду, що залишилась, що важливо для наступних побудов. Формально вказані функції виглядають так:

$$delO = *^2(S^3(=_Z, S^2(Od, \{(1,0)\}), S^2(Rn, S^2(\mathrm{E}_Z, I_1^1))), S^2(\Pi_Z, I_1^1)),$$

$$delZ = *^2(S^3(=_Z, \{(1,0)\}, S^2(Rn, S^2(\mathrm{E}_Z, I_1^1))), S^2(\Pi_Z, I_1^1)).$$

У визначенні *AscNames* також існує раніше не заданий предикат *GtNames*, що дозволяє порівняти два найменші закодовані імені у записах-аргументах, що є моделями векторів, які кодують деякий вихідний запис. Цей предикат можна визначити так:

$$GtNames = S^4(*^4(\Diamond(S^3(=_Z, S^2(Rn, S^2(\mathrm{E}, I_3^3)), \{(1,0)\}), S^3(=_Z, S^2(Rn, S^2(\mathrm{E}, I_2^3)), \{(1,0)\}), False), S^3(Neq, S^2(Rn, S^2(\mathrm{E}, S^2(\Pi, I_3^3))), \{(1,0)\}), S^2(\Pi, I_2^3), S^2(\Pi, I_3^3)), True, I_1^2, I_2^2).$$

Тепер сам предикат *isModel* представляється, наприклад, такою композицією чотирьох загаданих вище предикатів:

$$isModel = \Diamond(\Diamond(\Diamond(\Diamond(LastOne, FirstZero, False), Denotates10, False), StNames, False), AscNames, False).$$

Визначимо функцію *restore*, котра по моделі коду запису елемент за елементом відновлює вихідний запис. А саме, $restore = S^3(*^3(S^3(=_Z, \varnothing_Z, I_2^2), S^3(\nabla, I_1^2, ConvertPair), NextPair), \varnothing_Z, I_1^1)$, де *ConvertPair* – функція, котра відновлює по моделі коду запису перший елемент цього запису. Розглянемо цю функцію більш детально. Вона визначається через дві допоміжні функції. Одна, *convName*, перетворює послідовність нулів у ім'я елементу запису, інша, *convValue* – послідовність одиниць у денотат того ж елементу. Таким чином, $ConvertPair = S^3(convName, S^2(convValue, I_2^2), I_2^2)$, а

$convName = *^3(S^3(=_Z,\{(1,0)\},S^2(\mathrm{E}_Z,S^2(Rn,I_2^2))),S^2(Nm^+,I_1^2),S^2(\Pi_Z,I_2^2))$ та $convValue = S^2(\mathrm{E}_Z,S^2(Rn,S^3(Od^{-1},L,S^2(delZ,I_1^1))))$.

Отже, лема 3.1 повністю доведена. З цього випливає справедливість основного результату цього розділу.

Теорема. $\sigma_{Z^{(N,N)},\beta}$ – система утворюючих алгебри $A^{чр}_{Z^{(N,N)},\beta}$.

3.3 Семантичний аналіз мови Verilog

Verilog HDL (англ. Verilog Hardware Description Language) – це мова опису та розробки електронних цифрових систем. Розробники Verilog зробили його синтаксис дуже схожим на синтаксис мови C, що дещо спрощує його аналіз. Основні управляючі структури Verilog типу «if», «while» подібні конструкціям мови C. Хоча описи апаратури, написані на мові Verilog (як і на інших HDL-мовах), прийнято називати програмами, зауважимо, що, на відміну від загальноприйнятого поняття програми як послідовності інструкцій, тут програма задає структуру системи. Так само для мови Verilog не застосовують термін «виконання програми».

3.3.1 Загальні положення мови Verilog

Спочатку розглянемо синтаксис мови Verilog. Повна специфікація мови Verilog міститься у стандарті IEEE 1364. Тут будуть викладені тільки основні положення мови. Сама собою мова Verilog є мовою опису апаратного забезпечення. Апаратне забезпечення, яке описується, являє собою ієрархічну структуру модулів, на «входи» якої потрапляють сигнали, що змінюються з часом.

Для того, щоб отримати вичерпну інформацію про мову Verilog, можна розглянути його БНФ, яка описана в додатку А стандарту IEEE 1364.

Мову Verilog можна поділити на три множини:

- конструкції, що синтезуються;
- конструкції, що ігноруються під час синтезу»;

- конструкції, що не синтезуються.

Код, написаний виключно на конструкціях, що синтезуються або ігноруються гарантовано може бути відображений на ПЛІС(FPGA), на ASIC, на фотошаблони. Необхідно коротко згадати хоча б про основні обмеження мови щодо можливостей синтезу деяких синтаксичних конструкцій, насамперед керуючих структур. Для кожного синтезатора ці дані відрізняються, але їх більшість із них збігається. Цикли *forever* та *while* не синтезуються зовсім не синтезується. Цикл *repeat* синтезується лише тоді, коли вираз, який обчислює кількість ітерацій циклу може бути обчислений при синтезі. Цикл *for* синтезується тільки в тому разі, коли всі присвоєння для ітератору (індексу) циклу константні. Варто пам'ятати, зо синтезована цифрова схема не має ні часу початку роботи, ні часу закінчення роботи – вона просто існує. Тому ігноруються операції, які розробник задає для виконання на початку її роботи (але вони не ігноруються у симуляціях). Початкових значень регістрів також не існує. Крім того, ігноруються змінні, призначені для симуляції затримки виконання, котрі розробник може задавати для виразів. Вирішення цих «незручностей» знаходиться поза межами Verilog. Для таких випадків, наприклад, є сигнал скидування, про подачу якого та про реакцію на який має думати розробник. Саме сигнал скидування може бути умовним початком роботи схеми. Під час подачі тактового сигналу розробник задає рівномірний потік дискретного часу. Більшість цифрових електронних пристроїв працюють саме у дискретному часі, який задається тактовим сигналом. Такі пристрої називають синхронними.

3.3.2 Композиції у мові Verilog

У синтаксисі мови Verilog прихований набір композицій. Знайдемо композиційну семантику синтаксичних конструкцій, які використовуються у мові Verilog.

Почнемо розгляд композицій з *If*. У загальному вигляді структура галуження, *if*, може бути записана таким чином:

If(PredFunc, TrueFunc,FalseFunc),

де *PredFunc* – предикат, *TrueFunc* – функція, яка обчислюється, коли предикат істинний, *FalseFunc* – функція, яка обчислюється в іншому випадку.

Структура if може бути мати й іншу форму

If(PredFunc, TrueFunc, NullFunc)=if'(Predfunc,TrueFunc),

де *NullFunc* – «порожня функція», так звана null-функція, може уточнюватись по-різному, але фактично нічого не змінює. З точки зору цифрової схемотехніки функція нагадує мультиплексор (рис. 3.1).

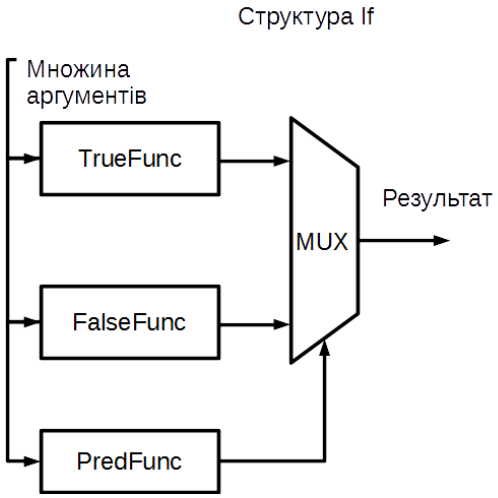


Рис. 3.1 Схематичне представлення композиції галуження

Одночасно обчислюються предикат, випадок для істинного предикату та випадок для хибного значення предикату. Результат предикату (істина чи хиба, 1 чи 0) передається на адресний вхід мультиплексору та залежно від його істинності сигналу мультиплексор подає на вихід данні з першого чи другого його входів. Операція галуження дуже важлива для забезпечення формальної системи повнотою по Тюрінгу.

Наступною важливою композицією є конструкція *case*, що може розглядатись як загальний випадок від *if*. При цьому вона має більше двох варіантів можливих значень. Записати її можна так

case(ValueFunc,DefaultFunc,Case1Func,Case2Func,…,CaseNFunc)

де *ValueFunc* – функція, що повертає значення, що аналізується, *Case1Func, Case2Func,…,CaseNFunc* – функції, результат який повертається залежно від

значень *ValueFunc*, 1,2,…,N, відповідно, *DefaultFunc* – функція, яка повертається у решті випадків.

Схемотехнічно ця композиція може бути зображена таким чином (рис. 3.2).

У цій композиції обчислюються всі можливі значення case, зокрема й значення по замовчуванню, а також величина, що визначає, яке з цих значень прийме *case*. Всі можливі варіанти передаються на мультиплексор. Обирається той із них, котрий відповідає обчисленій величині *ValueFunc*. У разі, якщо жоден із варіантів не відповідає цій величині, результатом стає значення *DefaultFunc*. Пристрій NEG аналізує те, чи заданий варіант для поточного значення *ValueFunc*; якщо такого варіанту не існує, то на вихід case комутується значення *DefaultFunc*. Такий пристрій легко організувати на основі комбінаційної логіки, але наразі це не є предметом розгляду.

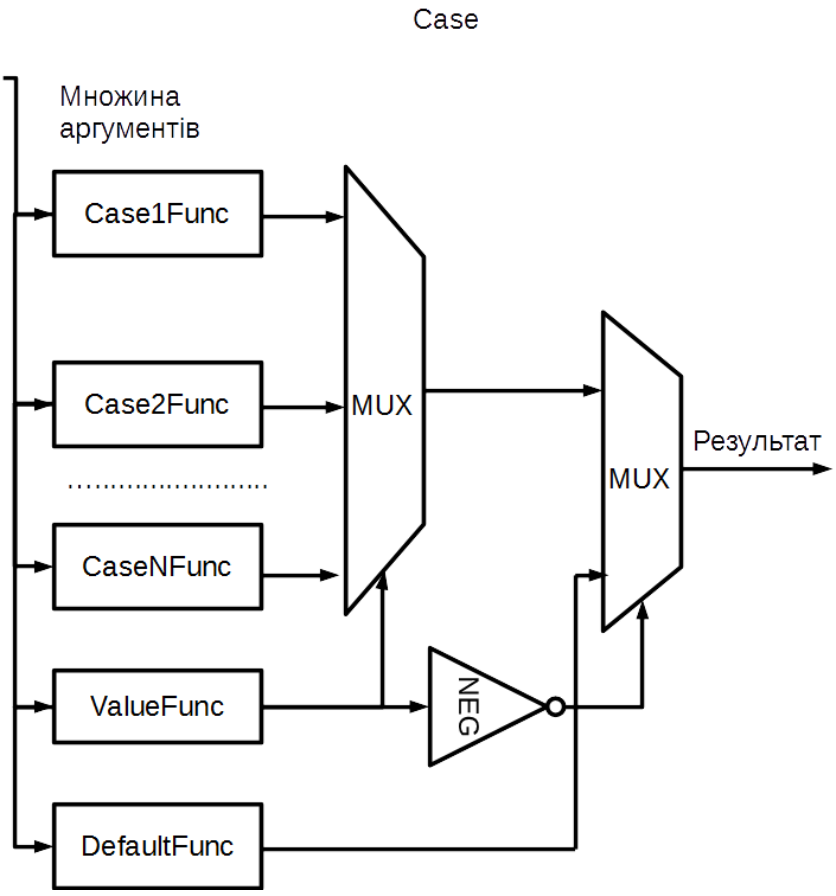


Рис. 3.2 Схематичне зображення композиції *case*.

Композиція *repeat* дозволяє застосовувати одну функцію (*AppFunc*) скінченну кількість разів (*N*) до аргументів.

repeatN(AppFunc)

Схематчино її можна записати як зображено на рис. 3.3.

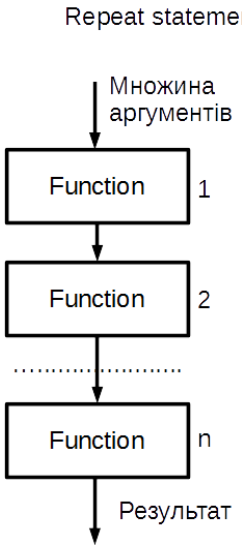


Рис. 3.3 Схемотехнічне представлення композиції *repeat*.

Ця композиція особлива ти, що приймає на вхід лише один аргумент-функцію, тим самим композиціюється сама з собою.

Наступна композиція – for. Вона подібна до інших структур for, що зустрічаються у різних мовах програмування. Якщо необхідно, щоб ця чи інша структура циклювання в мові Verilog могла бути такою, що може синтезуватись, необхідно, щоб кількість ітерацій циклу могла бути обчислена на етапі синтезу.

for(InitFunc,CondFunc,PostFunc,BodyFunc)

З урахуванням обмежень, які накладаються у разі, якщо треба досягти можливості її синтезу, її можна виродити в структуру на базі *repeat*. Функція *InitFunc* виконується до початку основного циклу, предикат виходу з циклу – *CondFunc*, тіло циклу *BodyFunc*, після кожної ітерації виконується *PostFunc*.

Композиція *while* має ті самі недоліки, що і *for*.

while(condFunc,bodyFunc)

Ані *BodyFunc*, ані *CondFunc* у разі потреби виконання умови синтезованості не повинні впливати на кількість ітерацій *while*, тим самим теж вироджуючись у *repeat*.

Хоча в Verilog управляючих структур багато, для них вдалось виділити лише дві композиції: галудження та повторення функції n раз, де n – константа.

Також у мові менш помітна наявна композиція суперпозиції (підстановки), яка обумовлена можливістю створення ієрархічних модулів. Наприклад:

```
module topmodule(IN0, IN1, OUT);
    input wire [7:0] IN0;
    input wire [7:0] IN1;
    output wire [7:0] OUT;
    wire [7:0] w1, w2,
    module1 m1(IN0, IN1, w1);
    module2 m2 (IN0, w1, w2);
    module3 m3 (IN0, w2, OUT);
endmodule
```

Модуль у мові Verilog можна представити як функцію. В коді у складі основного модуля topmodule існують три модулі m1, m2, m3. Останній порт у кожному з модулів є вихідним, а інші – вхідні. Провідники w1 та w2 дозволяють передати вихід одного модуля на вхід іншого. Таким чином, це явище можна розглянути як суперпозицію.

Суперпозицією можна також назвати послідовний запис команд у блоках *begin-end*, наприклад:

```
module (a, b, c);
input wire [7:0] a;
output reg [7:0] b;
input wire [7:0] c;
```

```
reg [7:0] d;
always @(posedge CLK) begin
    d = a + 5;
    b = d * c;
end
endmodule
```

Послідовні операції додавання та множення можуть бути розглянуті як функції. Результат операції додавання «+» далі використовується в операції множення «*». Таким чином, це можна розглядати як неявний випадок застосування композиції суперпозиції.

Водночас група названих композицій не повна за Тюрінгом, оскільки на ній можна побудувати тільки комбінаційні схеми. Послідовнісні схеми таким чином побудувати неможливо. Щоб подолати це обмеження, необхідно ввести поняття часу та станів, чого немає у написаному вище. Ці поняття виходять за межі мови Verilog.

Насправді задача отримання системи, повної за Тюрінгом, частково лежить на плечах розробника. Для повноти за Тюрінгом достатньо також розробити механізм циклювання, в якому кількість ітерацій не була б відома завчасно. Таку композицію можна отримати на мові Verilog за умови, що розробник гарантує наявність тактового сигналу на вході схеми для забезпечення протікання дискретного часу. Ось, наприклад, реалізація повнофункціональної композиції *while*.

```
module topmodule(CLK, ST, IN, OUT);
    input wire [7*3:0] IN;
    input wire CLK, ST;
    output reg [7*3:0] OUT;
    wire [7:0] mb_result;
    wire mc_result;
```

```
    reg [7*3:0] MI, MCI;
    wire [7*3:0] MR;
    module_condition mc(MCI, mc_result);
    module_body mb(MI, MR);
    always @(posedge CLK) begin
        if (ST == 1) begin
            MI <= IN;
            MCI <= IN;
        end else begin
            if (mc_result == 1) begin
                MI <= MR;
                MCI <= MR;
            end else begin
                OUT <= MI;
            end
        end
    end
endmodule
```

Це композиція двох функцій, які виражені модулями *mc* (предикат) та *mb* (тіло циклу). Для початку обчислень необхідно подати сигнал *ST*, за яким на наступний тактовий імпульс *CLK* вхідні аргументи передаються модулю предикату (регістр *MCI*) і модулю тіла циклу (*MI*). Поки вихід модуля умови *mc_result* рівний 1, тіло циклу приймає свій вихід *MR* на вхід *MI*. Після того, як предикат перестає бути істинним, результат із регістру *MI* видається на вихід модуля topmodule (*OUT*).

Використання композиційного підходу до конструювання з таким чином прихованими композиціями можливе, хоча й нетривіальне. Деякі композиції не мають своїх прообразів у мові Verilog.

У цьому підрозділі АПС досліджується зокрема з використанням ППА. Мотивом вибору ППА виступала саме її простота. Це важливо, адже об’єктом дослідження виступають не самі композиційні алгебри, а їхнє застосування в АПС. Звісно, якщо задачу можна вирішити в ППА, то цей є своєрідною гарантією того, що вона може бути вирішена більш потужними засобами. Так у цьому підрозділі досліджено ППА на деяких нових носіях, зокрема класах функцій над записами як над маніпулятивними, такими, що маніпулюють складовими елементами носія, не змінюючи їх, у нашому варіанті – запису, так і не маніпулятивними, які можуть змінювати складові елементи запису. Клас функцій над записами представляє великий практичний інтерес. Адже записи – популярна структура даних, яка лежить в основі key-value баз даних, хеш таблиць і т.д. Безперечно, такі дослідження додають цінності роботі.

Окрім зазначеного вище, досліджено семантичні суті синтаксичних конструкцій HDL-мов на прикладі мови Verilog та деякі інші їхні семантичні аспекти таких. Під час дослідження мови Verilog виділено основні синтаксичні конструкції. Їхні семантичні суті докладно описані та можуть розглядатись як композиції. Таким чином, викрито композиційну алгебру яка лежить в основі мови Verilog, що, зі свого боку, дозволяє застосовувати механізм АПС для вирішення задач із розробки апаратного забезпечення. В наступному підрозділі імплементовано адаптивне процесональне середовище, яке дозволяє отримувати рішення в Verilog HDL.

РОЗДІЛ 4. ІМПЛЕМЕНТАЦІЯ АДАПТИВНОГО ПРОЦЕСОНАЛЬНОГО СЕРЕДОВИЩА

4.1 Мови специфікації композицій

Під час розробки середовища розроблені графічна й текстова мови запису композицій. Вони дозволяють задавати та застосовувати функції, задавати та застосовувати композиції й застосовувати метакомпозиції. Записи в графічній мові відрізняються підвищеною сприйнятливістю.

Розглянемо БНФ текстової мови запису композицій. Ліва та права дужки, між якими містяться аргументи для аплікації функцій: *lFapply ::= <"("*> та *rFapply ::= <")"*>. Ліва та права дужки, між якими містяться аргументи для аплікації композицій*: lCapply ::= <"["*> та *rCapply ::= <"]"*>. Ліва та права дужки, між якими містяться аргументи для аплікації мета композицій: *lMapply ::= <"<"*> та *rMapply ::= <">"*>.

Операція іменування *assign ::= <"="*>. Ціле число *integer ::= <цілі числа*>. Ім’я name ::= < імена можуть складатися з цифр і літер, а починаються - завжди на літеру.

Аргументи функцій – *function_arguments ::= <integer>...* Аргументи композицій – *composition_arguments = <<Capply> | <name>>...* Аргументи метакомпозицій – *metacomp_arguments = <<Mapply> | <name>>...*

Застосування функцій *Fapply ::= <name><lFapply><function_arguments><rFapply>*, застосування композицій *Capply ::= <name><lCapply><composition_arguments><rCapply>,* застосування метакомпозицій *Mapply ::= <name><lMapply><metacomp_arguments><rMapply>*.

Іменування функцій, композицій, метакомпозицій: *valueAssignment ::= <name><assign><Fapply>, funcAssignment ::= <name><assign><Capply> compAssignment ::= <name><assign><Mapply>* .

Щодо графічного синтаксису, то його суть коротко можна виразити рисунком:

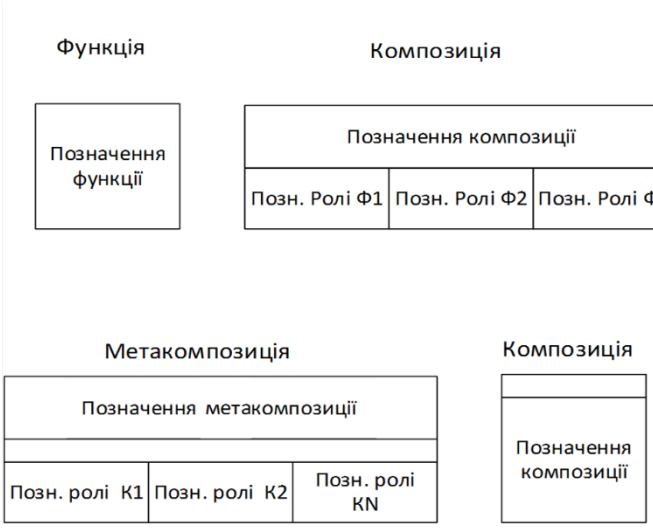


Рис. 4.1. Елементи графічного синтаксису для задавання функцій та композицій.

На рис. 4.1 зображені блоки, які являють собою функції, композиції та метакомпозиції. Усі вони можуть бути об’єднані в ненаправлений ациклічний граф (дерево). Ребра графа представляють зв’язки між композиціями, метакомпозиціями та їхніми аргументами. У роді позначень можна використовувати як ім’я відповідного елементу, так і схему його побудови, намальовану на місці для запису позначення.

Ось приклад запису програми для виділення автомобільних номерних пластин у текстовій мові специфікації композицій.

```
main = S[cvFindContours1,
    S[cvThresh1_64,
        S[cvBothat2,
            S[cvClose2,
                S[
                    cvGaussian1_21_1,
                    cvRgb2Gray1
                ],
                cvOnes1_1_13
            ],
            cvOnes1_1_13
```

```
            ]
        ]
    ]
```

За квадратними дужками знаходяться назви композицій. `cvFindContours1` – функція пошуку контурів, `cvThresh1_64` – функція порогу (64 з 256) для зображення, `cvBothat2` – функція “bothat” з маскою, `cvClose2` – функція замикання зображення з маскою, `cvGaussian1_21_1` – фільтр Гауса з радіусом 21 та σ=1, `cvOnes1_1_13` – функція створення матриці 1x13.

В результаті трансляції ІТ-рішення з МСК в цільовий синтаксис, отримано програму

```
std::vector<std::vector<Point> > cv_findcontours_9a15139a9(Mat arg1);
Mat cv_thresh_64_d0a820a75(Mat arg1);Mat cv_bothat_af5b7857b(Mat arg1, Mat
arg2);
Mat cv_close_9becf0c68(Mat arg1, Mat arg2);
Mat cv_gaussian_21_1_872f7aab5(Mat arg1);
Mat cv_rgb2gray_3c3ca47a9(Mat arg1);
Mat superposition_634f42c54(Mat arg1);
Mat cv_ones_1_13_530178292(Mat arg1);
Mat superposition_b62797068(Mat arg1);
Mat cv_ones_1_13_426f90f13(Mat arg1);
Mat superposition_63b2d46a0(Mat arg1);
Mat superposition_bdd1ed8e4(Mat arg1);
std::vector<std::vector<Point> > superposition_dc9a4186e(Mat arg1);

std::vector<std::vector<Point> > cv_findcontours_9a15139a9(Mat arg1) {
      std::vector<std::vector<Point> > contours;
      std::vector<cv::Vec4i> hierarchy;
      findContours(arg1.clone(), contours, hierarchy, cv::RETR_LIST,
      cv::CHAIN_APPROX_SIMPLE);
      return contours;
}

Mat cv_thresh_64_d0a820a75(Mat arg1) {
      Mat result;
      threshold(arg1, result, 64, 255, THRESH_BINARY);
      return result;
}

Mat cv_bothat_af5b7857b(Mat arg1, Mat arg2) {
      Mat result;
      morphologyEx(arg1, result, MORPH_BLACKHAT, arg1);
      return result;
}

Mat cv_close_9becf0c68(Mat arg1, Mat arg2) {
      Mat result;
      morphologyEx(arg1, result, MORPH_CLOSE, arg1);
      return result;
}

Mat cv_gaussian_21_1_872f7aab5(Mat arg1) {
```

```
	Mat blured;
	GaussianBlur(arg1, blured, cv::Size(21,21), 1);
	return blured;
}

Mat cv_rgb2gray_3c3ca47a9(Mat arg1) {
	Mat gray;
	cvtColor(arg1, gray, CV_BGR2GRAY);
	return gray;
}

Mat superposition_634f42c54(Mat arg1) {
	Mat retval;
	Mat result1;
	result1 = cv_rgb2gray_3c3ca47a9(arg1);
	retval = cv_gaussian_21_1_872f7aab5(result1);
	return retval;
}

Mat cv_ones_1_13_530178292(Mat arg1) {
	return Mat::ones(1, 13, CV_8U);
}

Mat superposition_b62797068(Mat arg1) {
	Mat retval;
	Mat result1;
	Mat result2;
	result1 = superposition_634f42c54(arg1);
	result2 = cv_ones_1_13_530178292(arg1);
	retval = cv_close_9becf0c68(result1, result2);
	return retval;
}

Mat cv_ones_1_13_426f90f13(Mat arg1) {
	return Mat::ones(1, 13, CV_8U);
}

Mat superposition_63b2d46a0(Mat arg1) {
	Mat retval;
	Mat result1;
	Mat result2;
	result1 = superposition_b62797068(arg1);
	result2 = cv_ones_1_13_426f90f13(arg1);
	retval = cv_bothat_af5b7857b(result1, result2);
	return retval;
}

Mat superposition_bdd1ed8e4(Mat arg1) {
	Mat retval;
	Mat result1;
	result1 = superposition_63b2d46a0(arg1);
	retval = cv_thresh_64_d0a820a75(result1);
	return retval;
}

std::vector<std::vector<Point> > superposition_dc9a4186e(Mat arg1) {
	std::vector<std::vector<Point> > retval;
	Mat result1;
	result1 = superposition_bdd1ed8e4(arg1);
	retval = cv_findcontours_9a15139a9(result1);
	return retval;
}
```

4.2 Архітектурні особливості реалізації АПС

В ході роботи над дисертацією розроблено дві версії дослідної реалізації АПС. Перша версія дослідної реалізації АПС була жорстко прив'язана до алгебри Чорча [11], що складалась з композицій суперпозиції, мінімізації, та примітивної рекурсії [67]. Підтримувалась мова побудови дескрипцій апаратного забезпечення Verilog, в якій можна було вирішувати нескладні задачі з розробки АтаПЗ. Вихідний код рішення доступний за посиланням https://github.com/player999/maltsev_verilog/tree/positional .

У другій версії дослідної реалізації АПС відпала необхідність жорсткого прив'язування до набору композицій завдяки гнучкій архітектурі та з'явилась можливість створювати нові композиції завдяки додаванню механізму метакомпозицій. Код рішення доступний за посиланням https://github.com/player999/AdapativeProgrammingEnvironment. На рис. 4.5 показана архітектура такого АПС.

Введення даних можливе як з графічного користувацького інтерфейсу так і через файл, ім'я якого здається аргументом командного рядка.

Допускається задання рішення як з використанням редукцій (рис. 4.2), які перетворюються в записи в мові специфікації композицій, так і напряму у мові специфікації комопзицій (рис. 4.3).

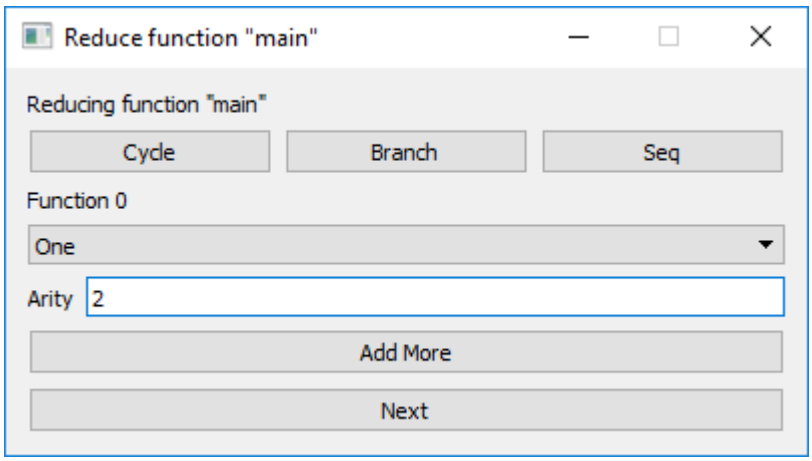


Рис. 4.2 Задання рішення з використанням редукцій.

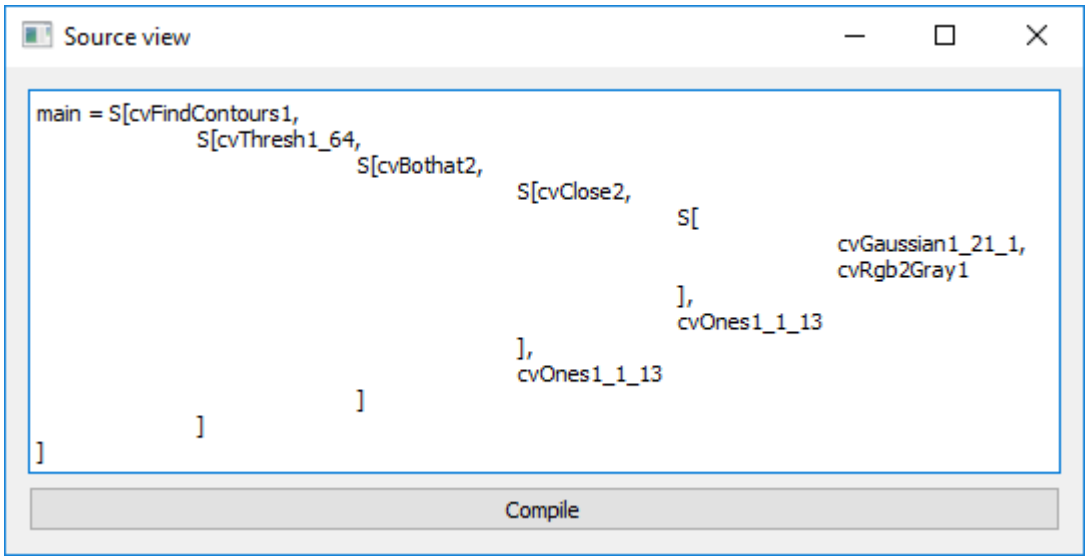


Рис. 4.3 Задання рішення в МСК.

Парсер будує абстрактні синтаксичні дерева виразів в МСК. Компонувальник компонує ці дерева в єдине АСТ.

Генератор цільового синтаксису компілює АСТ в цільовий синтаксис (рис. 4.4).

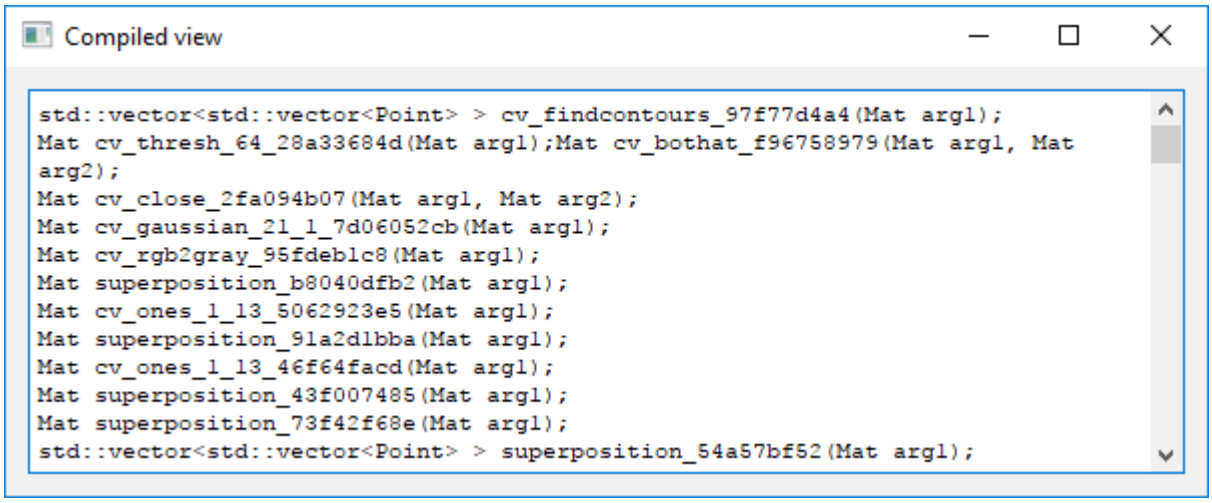


Рис. 4.4 Цільовий синтаксис рішення в мові С.

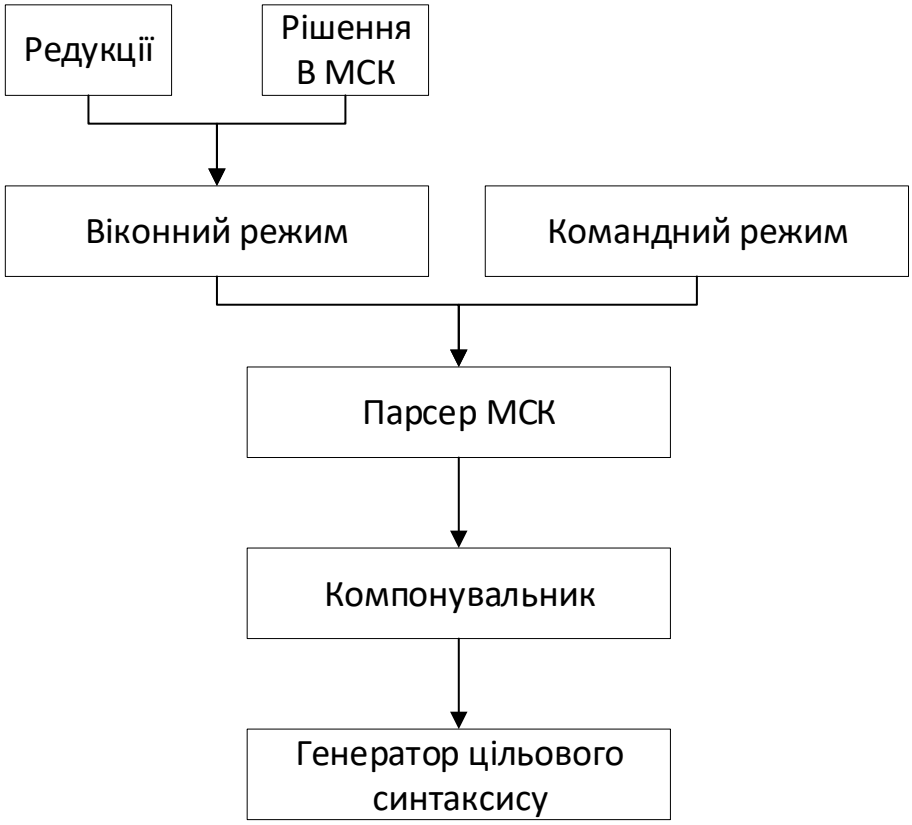


Рис. 4.5 Архітектура АПС

4.3 Інтеграція композиційного апаратного забезпечення з обчислювальними системами

4.3.1 Verilog модулі адаптивного апаратного забезпечення

Апаратне забезпечення, синтезоване з Verilog коду, наразі не може бути ефективно використане, не бувши інтегрованим із традиційними обчислювальними системами, зокрема комп'ютерами стандарту IBM PC, процесорними підсистемами систем на кристалі та ін. У зв'язку з цим постає задача інтеграції такого апаратного забезпечення з такими обчислювальними системами. У якості типового випадку реалізації такої інтеграції запропонована інтеграція з Altera Cyclone V, на якій виконується ОС на базі ядра Linux. Розглянемо детальніше питання взаємодії синтезованого Verilog апаратного забезпечення(АЗ) з обчислювальною системою.

Синтезований обчислювальний модуль має такий інтерфейс (рис. 4.6).

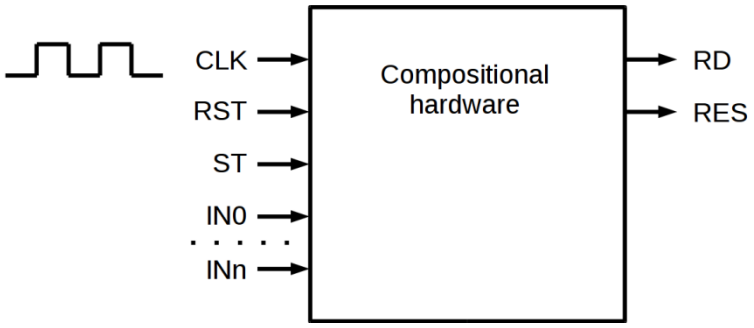


Рис. 4.6 Синтезований обчислювальний модуль.

- RST – сигнал скидування зупиняє усі обчислювання та очищує вихідний регістр;
- ST – «стартовий» сигнал, починає обчислення з початку, не зважаючи на внутрішній стан модуля, очікується, що під час його подавання на вході встановлені дійсні IN0…IN1.
- CLK – тактовий сигнал;
- RD – сигнал готовності, вказує на те, що вихідний регістр модуля містить дійсні значення;
- RES – вихідний регістр модуля;
- IN1…INn – аргументи модуля.

Природно, що ширина портів RES та IN1…INn однакова.

Задачу інтеграції обчислювального модуля з традиційними обчислювальними системами може бути розкладена на два кроки:

1. Постановка модуля на шину, як найпопулярніший спосіб взаємодії традиційних обчислювальних ядер із периферійними пристроями.
2. Написання програмного забезпечення та драйверів для взаємодії з обчислювальними модулями, розташованими на шині.

Популярні шини (Wishbone, Avalon, Amba та ін.) є приблизно еквівалентними за своєю структурою. Розглянемо шину, на якій проводились дослідні роботи – Avalon (реалізація memory mapped – пристрою). Ядром шини є система міжз’єднань (bus fabric), яка включає засоби комутації та логіку комутації пристроїв на шині між собою, зокрема так звані «арбітри». Вони

дозволяють визначати послідовність доступу пристроїв до шини, оскільки одночасно на шині тільки один пристрій може займати канал передачі даних. Крім того, у структурі шини існують два види інтерфейсів пристроїв: master та slave. Master відрізняється від slave тим, що він може ініціювати транзакції на шині та «звертатись» до slave'ів. Slave може лише генерувати відповідь для master'а та ініціювати транзакції лише опосередковано: через переривання, які власне не відносяться до шинної технології.

Логічно, що найкращою роллю для пристрою, який створений адаптивним методом розробки є роль slave-пристрою. Натомість постає задача автоматизації такої розробки. Для цього розроблено програмне забезпечення, яке генерує slave-пристрій для САПР Altera Quartus II, котрий можна застосовувати в програмі Qsys із програмного комплексу Quartus II. Згенерований таким ПЗ пристрій матиме наступні порти:

- RST_I – сигнал скидання;
- CLK_I – тактовий сигнал;
- WE_I – сигнал запису;
- STB_I – строб, сигнал вибору мікросхеми або chipselect;
- ADR_I – порт адреси;
- DAT_I – вхідні дані (writedata);
- DAT_O – вихідні дані (readdata);
- IRQ_O – сигнал переривання.

Варто згадати деякі особливості у цьому наборі портів. Насамперед це відсутність сигналу зчитування, але без нього можна обійтись, оскільки єдині дані, що зчитуються з пристрою – це результат. Наступною особливістю є відсутність сигналу waitrequest, але це припустимо, оскільки єдиним master-пристроєм на цій шині буде процесорний модуль. Відсутній також сигнал byteenable, бо об'єм усіх транзакцій на шині кратний одному слову у нашому випадку. Додано сигнал переривання IRQ_O, за допомогою якого пристрій може сигналізувати процесору про завершення обчислень.

Звісно, сам собою цей пристрій приносить мало користі. Варто розглянути те, яким чином його можна інтегрувати, скажімо, в систему на кристалі. Розглянемо типову систему на кристалі з інтегрованим прискорювачем обчислень на прикладі SoC серії Altera Cyclone V із вбудованим ARM ядром, так званою апаратною процесорною системою (Hardware Processor System, HPS).

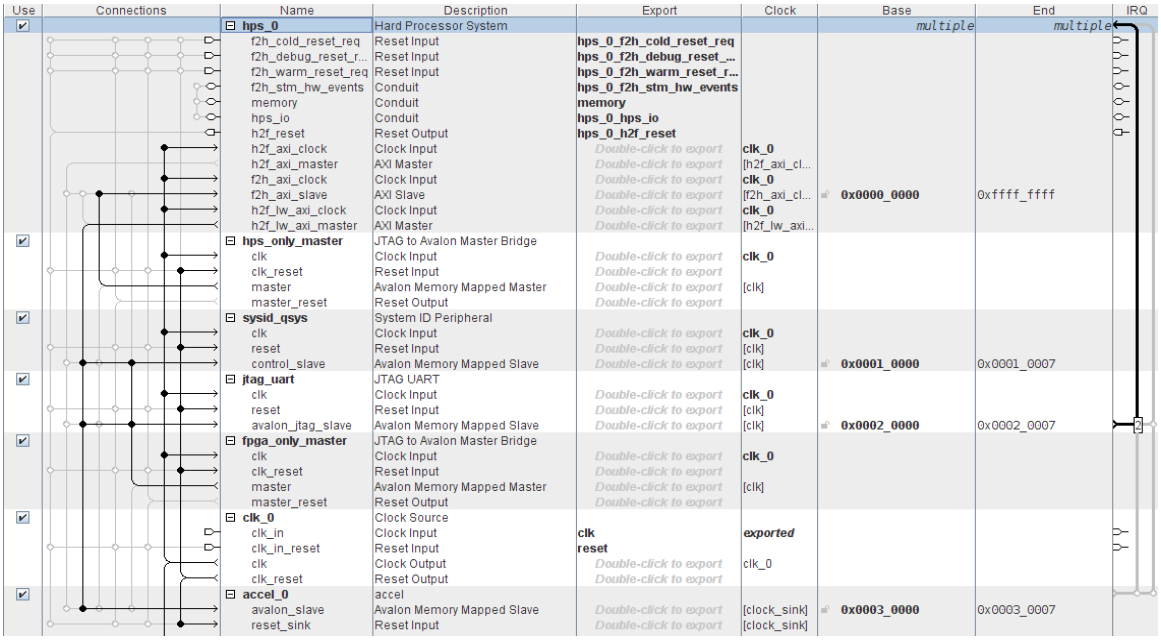


Рис. 4.7 Приклад системи на кристалі з прискорювачем.

HPS із процесорними ядрами ARM Cortex-A9 пов’язана з FPGA через AXI мости FPGA-to-HPS та HPS-to-FPGA, а також їхні «полегшені». Таким чином, master та slave пристрої можуть розміщатись як на FPGA, так і на процесорній системі. Окрім самого прискорювача обчислень у системі на кристалі (рис. 4.7), розташовані інші необхідні функціональні блоки.

- hps_0 – апаратна процесорна система (HPS);
- hps_only_master – міст з інтерфейсу налагодження JTAG на шину Avalon для налагодження роботи процесора;
- sysid_qsys – модуль ідентифікації системи на кристалі;
- fpga_only_master – міст з інтерфейсу налагодження на шину Avalon для налагодження пристроїв на FPGA;

- jtag_uart – перетворювач інтерфейсів JTAG <->UART для консольного доступу до HPS через послідовний інтерфейс;
- clk_0 – джерело тактових імпульсів та сигналу скидання;
- accel_0 – безпосередньо розроблений пристрій-прискорювач обчислень.

В роботі як приклад наведено пристрій для множення двох чисел. Його часові діаграми наведені на рис. 4.8.

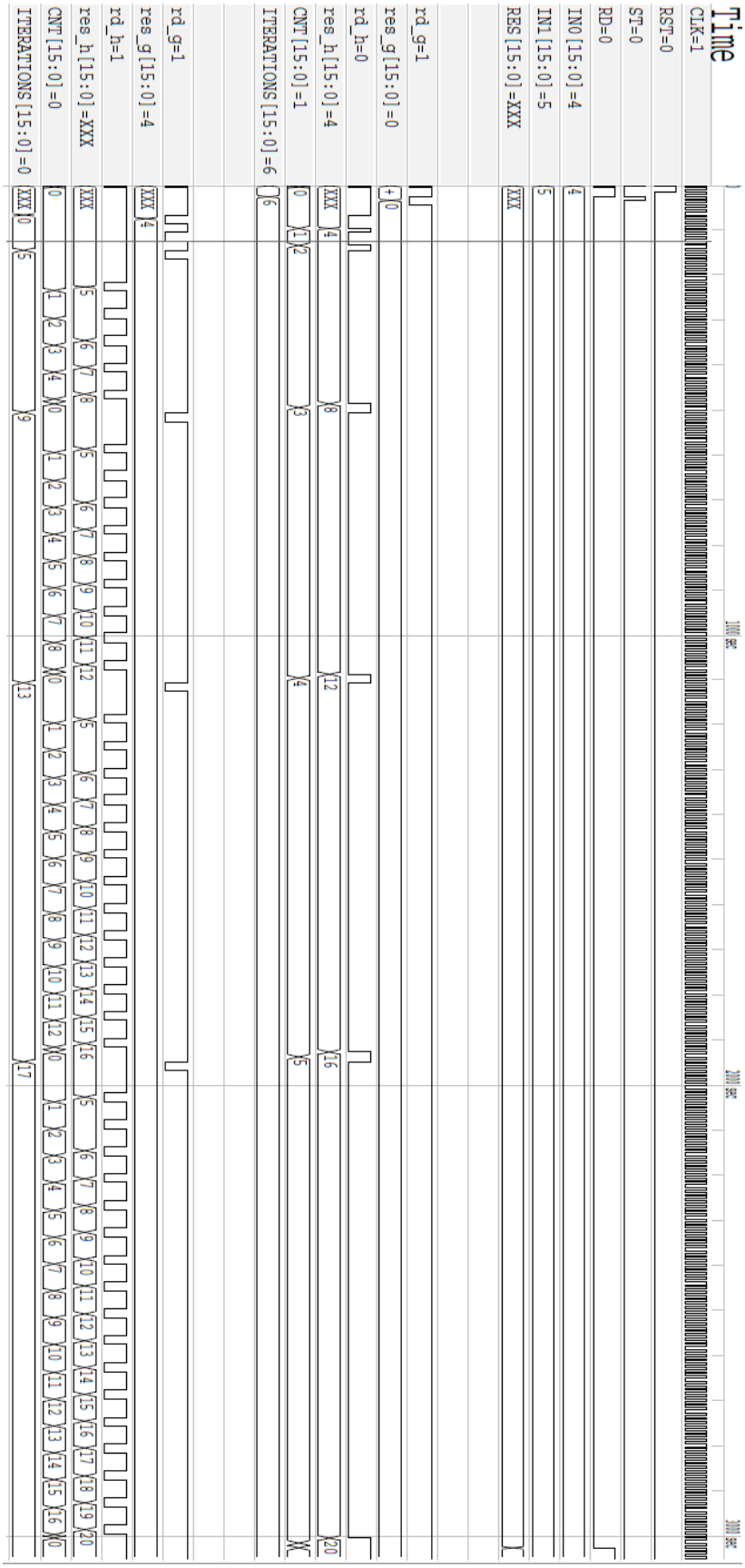


Рис. 4.8 Часові діаграми адаптивного апаратного забезпечення для множення.

4.3.2 Драйвери для адаптивного апаратного забезпечення

Залишилось розглянути, як такий пристрій буде виглядати в очах програміста, що його використовує, яка його програмістська модель.

Пристрій являє собою набір регістрів у пам’яті. Кожен регістр під час запису – один із аргументів функції, яка реалізована у рішенні. При зчитуванні будь-якого з регістрів зчитується значення функції, яка реалізована в рішенні.

Є два варіанти отримати доступ до функції з операційної системи (у якості прикладу розглядаємо GNU/Linux) – написати модуль ядра, який би міг працювати у реальному режимі та зчитувати та записувати дані фізичної пам'яті, представляючи на віртуальній файловій системи sysfs пристрій або використовуючи системні функції, які дозволяють відображати фізичну пам’ять у віртуальний простір адресу користувацької програми (user-space).

Спочатку розглянемо другий варіант як той, що простіший. Для того, щоб отримати доступ до конкретного діапазону віртуальних адрес (наприклад, де знаходиться прискорювач), необхідно використати пристрій /dev/mem, робота з яким нагадує роботу з файлом. Функцією open() отримується дескриптор для роботи з /dev/mem. При використанні цього дескриптора здійснюється відображення заданого фізичного діапазону пам’яті у віртуальний адресний простір процесу за допомогою функції mmap(). Таким чином, процес з user-space може працювати безпосередньо з memory-mapped пристроями.

Тепер розглянемо задачу взаємодії між операційною системою та пристроєм через модулі ядра (драйвери). У модулях ядра можна без проблем писати код, який використовує фізичний простір адрес. Модулі ядра бувають вбудовані і завантажувані (Loadable kernel module, LKM). Модуль ядра – об’єктний файл, який містить код, що розширює функціональність ядра операційної системи. У випадку з LKM цей код може завантажуватися і вивантажуватись динамічно, залежно від необхідності. Іншим питанням, на яке необхідно відповісти, є те, як код у ядрі буде взаємодіяти з програмами, що виконуються у віртуальному просторі адрес (userspace). Є дві відповіді на це питання: системні виклики та віртуальна файлова система. Перший варіант є

небажаним, оскільки додавання нових системних викликів у більшості проводиться розробниками Linux, а використаний код виклику може бути потім використаний розробниками Linux.

Таким чином, оптимальним вирішенням проблеми взаємодії коду в ядрі з userspace є sysfs. Sysfs – віртуальна файлова система, яка створюється ядром Linux. Використовуючи віртуальні, sysfs експортує інформацію про версії різних підсистем ядра, апаратні пристрої та пов’язані з ними драйвери в userspace. До того ж, окрім інформування користувачів та розробників ПЗ, sysfs використовується для конфігурування пристроїв і підсистем ядра. Користувацька модель адаптивного апаратного забезпечення в sysfs містить n+1 файл, кожен із n файлів відповідає за один вхід та n+1'ий файл відповідає регіструю пристрою з результатом. Вони нумеруються наступним чином in1, in2, in3,...inn, res (рис. 4.6). Кожен файл «in» доступний тільки для запису, а «res» – тільки для зчитування. Обчислення запускаються після того, як записаний останній вхідний регістр (inn). Усі ці файли розміщені в */sys/kernel/mm/devicename* (рис. 4.9).

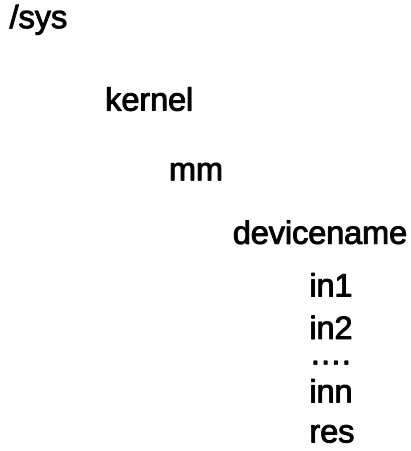


Рис. 4.9 Представлення адаптивного апаратного забезпечення в sysfs.

Розглянемо інтерфейс програміста (API) ядра для створення такого драйверу.

У розділі описано практичне застосування концепції адаптивних процесональних середовищ для розробки засобів розробки, що адекватно підтримують активну роль суб’єкта в процесі розробки. За результатами роботи

розроблено адаптивне процесональне середовище, яке дозволяє застосовувати та породжувати композиції та здійснювати декомпозицію програми. Воно підтримує автоматичну генерацію коду програми в мовах Сі та Verilog HDL, а також має графічний інтерфейс для декомпозиції задачі.

Під час розробки середовища розроблені графічна та текстова мови запису композицій. Вони дозволяють задавати і застосовувати функції, задавати та застосовувати композиції, а також застосовувати метакомпозиції. Записи в графічній мові відрізняються підвищеною сприйнятливістю.

Звісно, таке адаптивне процесональне середовище було б складно застосовувати без можливості інтеграції отриманих рішень в обчислювальні системи, що вже існують. Особливо це стосується апаратного забезпечення. Саме тому описана технологія розробки драйверів для апаратного забезпечення, отриманого за допомогою адаптивного процесонального середовища (адаптивного апаратного забезпечення, ААЗ) для операційної системи на базі ядра Linux. Є два варіанти отримати доступ до ААЗ із операційної системи: написати модуль ядра, який міг би працювати у режимі супервізора та зчитувати й записувати дані фізичної пам'яті, представляючи на спеціальній файловій системі sysfs пристрій; або використати системні функції, які дозволяють відображати фізичну пам’ять у віртуальний простір адрес користувацької програми (user-space).

Ми зупинились на першому як на найбільш широковживаному й такому, що найкраще відповідає поставленій задачі, але у розділі розглянуто обидва способи.

4.4 Аналіз ефективності АПС

Підвищення ефективності розробки з використанням АПС обґрунтоване, але його кількісні показники ще не визначені. Існує ряд методів для оцінки його ефективності, зокрема: COCOMO, SLIM, PERT та ін. Серед них найвідомішим є COCOMO (Constructive Cost Model), запропоновану Барі Боемом [108], що і був

використаний для оцінки використання АПС. Затрати на розробку попередньо оцінюються за наступною формулою

$$E_{пот} = A*(Size)^B,$$

де $E_{пот}$ -- затрати на розробку в людино-місяцях, $Size$ -- відносний розмір рішення, що залежить від кількості рядків у рішенні, мови програмування, функціоналу, що реалізовується, A -- константа, в залежності від типу рішення, B -- масштабуючий показник.

Ця формула приводиться для попередньої оцінки рішення. Параметри формули визначаються емпірично. На константу A впливає, зокрема тип рішення: органічний(маленька команда та невисокі вимоги до рішення), напівроздільний (середня команда зі змішаним досвідом та змішані вимогами до рішення), вбудований (велика команда та жорсткі вимоги до рішення). Ці обставини відносяться до прагматичних умов, на них впливати неможливо, тому залишаємо їх поза розглядами. Константа має значення в середньому ~2 та може змінюватись в широких межах [109]. Розмір рішення – теж об'єктивний параметр та залежить від методів розробки. Масштабуючий показник B враховує специфіку рішення та визначається наступною формулою

$$B = 1.01 + \sum_{j=1}^{5} SF_j,$$

де SF_j -- масштабуючі константи. В таблиці зображено перелік таких констант. Кожен з цих коефіцієнтів визначається властивостями розробки та має значення, яке суб'єктивно визначається експертом градаціями від Very Low до Extra High. Таким чином для кожної властивості розробки маємо наступні рівні значимості: Very Low (Дуже низька), Low (Низька), Normal (Звичайна), High (Висока), Very High(Дуже висока), Extra High (Надзвичайно висока). Приведемо ці властивості та значення констант для кожного рівня значимості.

Таблиця 4.1 Константи для попередньої оцінки [109]

Поз.	Скороч.	Опис	VL	L	N	H	VH	EH
SF1	PREC	Розуміння продукту та технології організацією	0.05	0.04	0.03	0.02	0.01	0.00
SF2	**FLEX**	**Гнучкість розробки**	**0.05**	**0.04**	**0.03**	**0.02**	**0.01**	**0.00**
SF3	RESL	Ризики, розуміння архітектури	0.05	0.04	0.03	0.02	0.01	0.00
SF4	TEAM	Гармонічність колективу	0.05	0.04	0.03	0.02	0.01	0.00
SF5	**PMAT**	**Надійність процесу розробки**	**0.05**	**0.04**	**0.03**	**0.02**	**0.01**	**0.00**

АПС має очевидний значний вплив на гнучкість розробки (FLEX) та надійність чи зрілість процесу розробки (PMAT), таким чином ми можемо актуалізувати ці коефіцієнти рівними 0, що відобразить властивості АПС. Виходячи з цього максимальне значення B стає рівним не 1.26, а 1.16. Підсумовуючи сказане зобразимо графік максимально можливої економії затрат на розробку в залежності від розміру рішення. Цей графік розраховується по формулі

$$Economy = \frac{Size^{1.26} - Size^{1.16}}{Size^{1.26}} \cdot 100\%$$

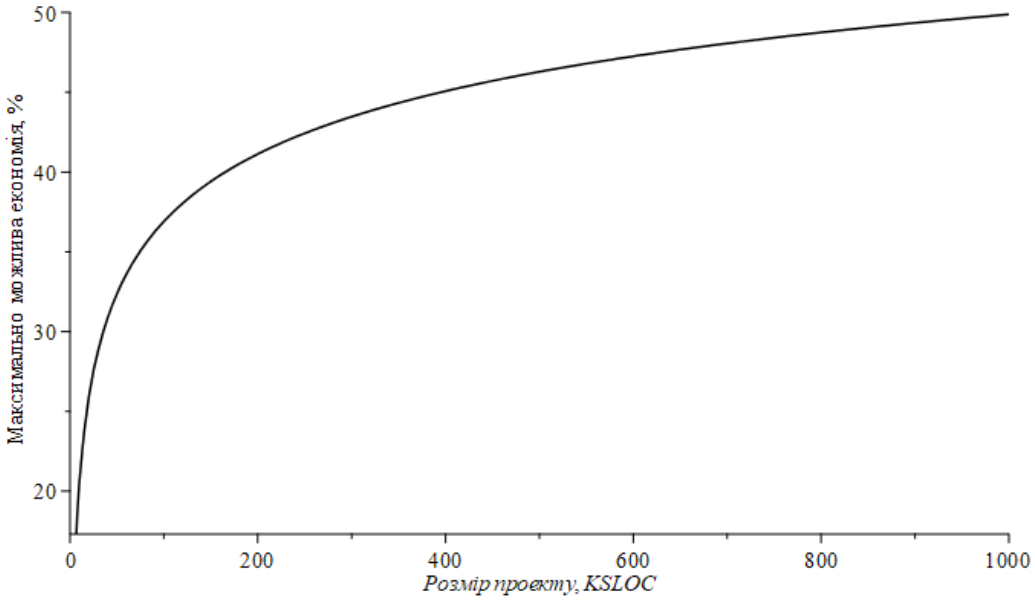


Рис. 4.10 Максимально можлива економія затрат на розробку

Як видно з графіку (рис. 4.10), чим більше розмір рішення (вимірюється в KSLOC – тисячах рядків коду), тим значніша економія.

Після початкових фаз розробки, затрати на розробку можна уточнити рядом з 17 корегуючих коефіцієнтів:

$$E_{adj} = E_{nom} \cdot \prod_{j=1}^{17} EM_j$$

Опис коефіцієнтів містить таблиця 4.2.

Таблиця 4.2 Корегуючі коефіцієнти.

Поз.	**Абр.**	**Опис**
EM1	**RELY**	**Необхідна надійність рішення**
EM2	DATA	Розмір бази даних за необхідності
EM3	CPLX	Складність продукту
EM4	**RUSE**	**Вимоги до можливості повторного використання**
EM5	**DOCU**	**Вимоги до документації**
EM6	**TIME**	**Вимоги до тривалості неперервної роботи**
EM7	STOR	Вимоги до пам'яті
EM8	PVOL	Змінюваність (волатильність) платформи
EM9	ACAP	Аналітичні здібності розробників
EM10	PCAP	Досвід командної розробки, комунікабельність розробників
EM11	AEXP	Досвід роботи
EM12	PEXP	Досвід розробки на даній платформі
EM13	LTEX	Знання мови розробки та інструментарію
EM14	PCON	«Текучість» розробників
EM15	**TOOL**	**Ефект інструментів розробки**
EM16	SITE	Рівень географічного розподілу команди
EM17	SCED	Вимоги щодо дотримання графіку розробки

Кожен з цих коефіцієнтів, як і складові показнику B ранжується від VL до EH. Таблиця значень цих коефіцієнтів приведена далі.

Таблиця 4.3 Значення корегуючих коефіцієнтів [109]

	VL	**L**	**N**	**H**	**VH**	**EH**
RELY	**0.75**	0.88	1	1.15	1.39	
DATA		0.93	1	1.09	1.19	
CPLX	0.75	0.88	1	1.15	1.30	1.66
RUSE		**0.91**	1	1.14	1.29	1.49
DOCU	**0.89**	**0.95**	1	1.06	1.13	
TIME			**1**	1.11	1.31	1.67
STOR			1	1.06	1.21	1.57
PVOL		0.87	1	1.15	1.30	
ACAP	1.5	1.22	1	0.83	0.67	
PCAP	1.37	1.16	1	0.87	0.74	
PCON	1.24	1.10	1	0.92	0.84	
AEXP	1.22	1.10	1	0.89	0.81	
PEXP	1.25	1.12	1	0.88	0.81	
LTEX	1.22	1.10	1	0.91	0.84	
TOOL	1.24	1.12	1	0.86	**0.72**	
SITE	1.25	1.10	1	0.92	0.84	0.78
SCED	1.29	1.00	1	1.00	1.00	

У зв’язку з тим, що АПС забезпечує гарантоздатність рішення, можна стверджувати, що додаткових зусиль для її забезпечення докладати не треба, що дозволяє зафіксувати RELY та TIME на рівнях VL та N відповідно. Крім того АПС забезпечує можливість повторного використання рішення, що теж дає можливість зафіксувати коефіцієнт RUSE на рівні L. Той факт, що проектна документація може бути автоматично згенерована, а від розробників необхідна

тільки користувацька документація, дозволяє зафіксувати коефіцієнт DOCU на рівнях VL та L. У зв'язку з тим, що наш інструмент розробки – АПС зручний для користувача та під нього підлаштовується, можна зафіксувати коефіцієнт TOOL на рівні 0.75. Сумарно ефект всіх актуалізацій коефіцієнтів рівний

$$C_{АСП} = RELY \cdot RUSE \cdot DOCU \cdot TIME \cdot TOOL = 0.75 \cdot 0.91 \cdot 0.89 \cdot 1 \cdot 0.72 \approx 0,4373$$

Що складає близько 56% економії затрат. Якщо ж врахувати ще економію, що забезпечує показник то можна оновити графік на рис. 4.10, то отримаємо

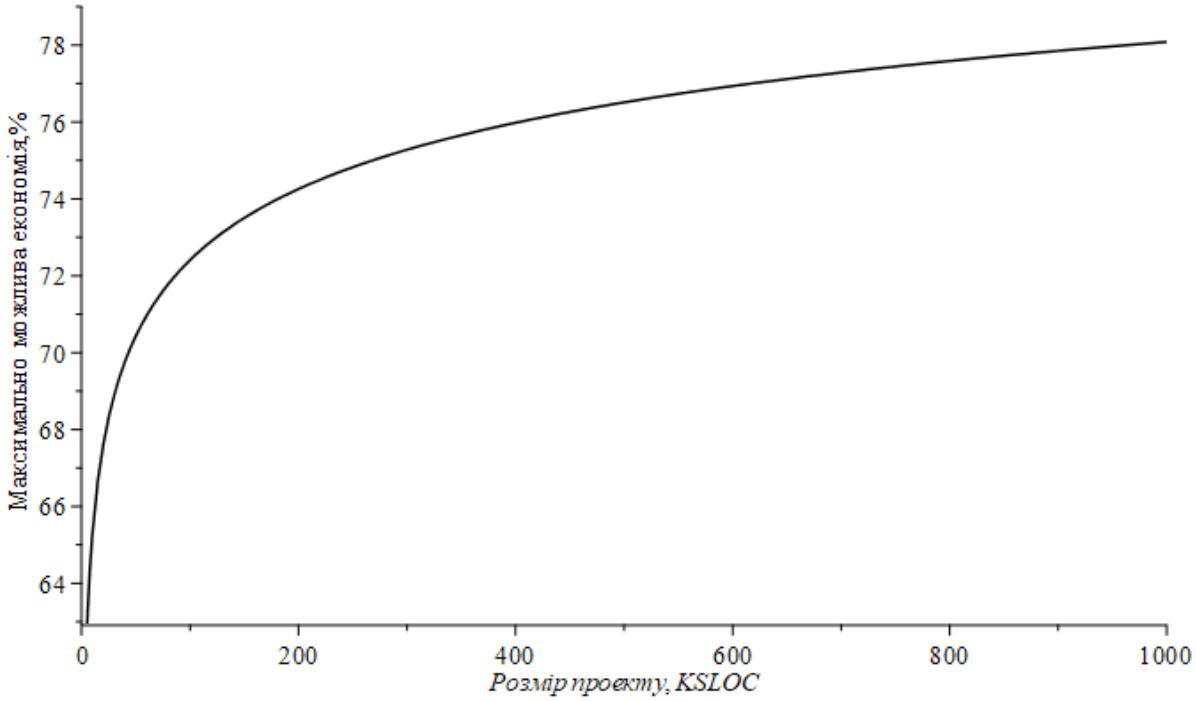


Рис. 4.11. Максимально можлива економія затрат на розробку з урахуванням з уточнюючих коефіцієнтів

Відповідно оновлюється і формула, за якою побудований графік

$$Economy = \frac{Size^{1.26} - Size^{1.16} \cdot 0,4373}{Size^{1.26}} \cdot 100\%$$

Виходячи з наведених вище знаходжень можна з високою впевненістю стверджувати, що АПС здатне забезпечити до 78% економії затрат на розробку рішення. Для більш точних оцінок необхідно застосувати модель оцінювання затрат COCOMO у повній мірі.

У розділі застосовані теоретичні засади АПС для семантичних досліджень АПС, зокрема сформульовано метод побудови породжуючих систем

для різних класів обчислюваних функцій, який був застосований для побудови породжуючих систем практично важливих класів функцій. Крім цього проведений семантичний аналіз мови Verilog, в результаті якого, вдалося виділити набір композицій, який лежить в основі мови. Розроблено дослідну реалізацію АПС, а в рамках розробки дослідної реалізації створені графічна та текстові мови специфікації композицій і засіб інтеграції АПС з традиційними обчислювальними системами. Крім того проаналізовано ефективність АПС.

ВИСНОВКИ

У дисертації розглянуто концептологічні засади АПС як ефективного інструменту створення якісного й надійного програмного та апаратного забезпечення.

Основним результатом дисертації є розроблений адаптивний підхід до побудови прагматико-обумовлених концептологічних специфікацій програмного та апаратного забезпечення. Автором запропоновано та обґрунтовану концептуально єдину понятійну систему АПС і на її основі створено відкрито-замкнені засади АПС. Основний акцент зроблено на врахуванні активної ролі суб'єкта у ІТД як її носія. Нееклектичне залучення ІТД до розглядів дало можливість коректно ставити питання релятивізації ІТД, врахування притаманних суб'єкту особливостей вирішення ІТ-задач. Це дозволило реально ставити та вирішувати питання підвищення ефективності створення та якості отримуваних ІТ-рішень.

За результатами проведеної змістовної релятивізації побудована логіко-предметна понятійна система АПС, основу якої складає загальне поняття композиції як уточнення змістовного поняття генетичної структури. Акцентування уваги на засадничій ролі композиції у ІТД також важлива через те, що навіть сьогодні залучення до розглядів композицій часто проводиться несвідомо та досить обмежено, а іноді й не проводиться взагалі. Так у різних мовах програмування чи HDL обмежуються м'якими зауваженнями щодо властивих їм т.з. «управляючих» синтаксичних структур. Про семантичний рівень і про підлеглість синтаксису семантиці мова у цьому випадку навіть не йде. Семантика тут задається традиційною інтерпретацією за синтаксичними структурами. Залучення до розгляду композицій дозволяє адекватно підтримати семантико-синтаксичну точку зору на ІТД та забезпечує реальний механізм реалізації конкретних середовищ ІТД. Таким чином були сформовані композиційні засади АПС.

На основі цих засад як концептуально єдиного носія породження та застосування композицій створені композитосутнісні засади моделей АПС. Показано, що АПС зводиться до ІТ-релятивізації універсального середовища

інтеграції (VCI). Це дозволяє коректно ставити та вирішувати питання ефективності ІТД та якості отримуваних результатів.

Логічним кроком в подальшому дослідженні стала прагматико-обумовлена типізація композицій, як один з способів розкриття суті композиції. Крім того розроблено метод композитосутнісної релятивізації, що створив основу для практичного застосування АПС.

У якості предметного замикання логіки АПС досліджено семантичні характеристики репрезентативних класів обчислюваних функцій над зліченними носіями, зокрема класом функцій та предикатів над записами. Результати отримано по алгебрам функцій над іменними множинами. Також досліджено семантичні структури, що містяться в мові Verilog HDL. У синтаксисі Verilog HDL вдалося виділити синтаксичні структури, які можуть бути проекціями композицій на синтаксис. Таким чином, відкривається можливість застосування композиційного підходу для розробки апаратного забезпечення.

Розроблено дослідну реалізацію АПС та підтримано можливість задавати композиції як у текстовому, так і в діалоговому режимі відповідно до суб'єктивного бачення вирішення задачі розробником. Базові набори функцій та композицій легко замінюються. Вирішено ряд репрезентативних задач, які демонструють ефективну розробку з використанням АПС. Крім того, вирішена задача інтеграції "композиційного" апаратного забезпечення з традиційними обчислювальними системами, а саме розроблено ПЗ для конвертації композиційного АЗ в IP-блок для САПР Quartus II та ПЗ для генерації програмного коду для драйверів Linux.

Подальші дослідження можуть включати в себе автоматичну оптимізацію композиційних конструкцій, віднаходження нових композиційних алгебр, зокрема в класах функцій над композитними носіями. На подальші дослідження також заслуговує розвиток технології інтеграції композиційного апаратного забезпечення до існуючих обчислювальних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] T. L. Zakharchenko, "Adaptive hardware design," *Технологический аудит и резервы производства,* no. 5, pp. 23-31, 2015.

[2] T. Zakharchenko, "Pragmatics dependent hardware design," in *Proceedings of the 2014 IEEE 34th International Scientific Conference "Electronics and nanotechnology"*, Kyiv, 2014.

[3] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» *Вісник КНУТД,* № 1, pp. 35-39, 2014.

[4] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» в *Збірник статей VI Міжнародної науково-технічної конференції молодих вчених «Електроніка-2013»*, Київ, 2013.

[5] Т. Л. Захарченко, «Вирішення проблеми повноти в композиційному програмуванні,» в *Матеріали XIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики»*, Київ, 2015.

[6] Т. Л. Захарченко, «Швидкодіючий багатошаровий персептрон, що гнучко масштабується». Україна Патент 72313, 10 08 2012.

[7] Т. Л. Захарченко, І. В. Редько, Д. І. Редько та П. О. Яганов, «Примітивна програмна алгебра обчислюваних функцій над записами,» *Наукові вісті НТУУ «КПІ»,* № 2, pp. 29-40, 2015.

[8] D. I. Redko, I. V. Redko, P. O. Yahanov and T. L. Zakharchenko, "Compositional basis in programmer activity," *Системні дослідження та інформаційні технології,* no. 4, pp. 83-96, 2015.

[9] Т. Л. Захарченко та І. В. Редько, «Проблема повноти в класі функцій над записами, які зберігають денотати,» *Наукові вісті НТУУ «КПІ»,* № 5, pp. 23-30, 2015.

[10] І. В. Редько, Д. І. Редько та Т. Л. Захарченко, Концептологічні основи проектування, Київ: Компринт, 2016.

[11] И. А. Басараб, Н. С. Никитченко и В. Н. Редько, Композиционные базы данных, Київ: Либідь, 1992, p. 192.

[12] В. Н. Редько, «Композиционная структура программологии,» *Кибенетика и системный анализ,* № 4, pp. 47-66, 1998.

[13] В. Н. Редько, «Универсальные программные логики и их применение,» в *Тезисы докладов IV всесоюзного симпозиума*, Кишинев, 1983.

[14] Д. Бекус, «Алгебра функциональных программ: мышление функционального уровня, линейные уравнения и обобщенные определения,» в *Сб. статей: Математическая логика в программировании*, Москва, Мир, 1991, pp. 8-53.

[15] А. Черч, Введение в математическую логику, Москва: ИИЛ, 1960, p. 485.

[16] J. McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I," *Communications of the ACM,* vol. 3, no. 4, pp. 184-195, 1960.

[17] Д. Робинсон, "Логическое программирование - прошлое, настоящее и будущее," in *Сбоорник статей "Логическое программирование"*, Москва, 1988, pp. 7-26.

[18] И. В. Редько, «Теория дескриптивных сред и ее применения,» Киев, 2008.

[19] В. Н. Редько и И. В. Редько, «Дескриптивные системы: ретроспективы и перспективы,» в *Вісник Київського національного університету ім. Т.Шевченка. Сер. Фіз.-мат.науки*, 2004, pp. 68-75.

[20] В. Н. Редько и И. В. Редько, «Дескриптологические основания информационных технологий,» *Кибернетика и системный анализ,* № 5, pp. 12-28, 2007.

[21] И. В. Редько, «Экспликативное моделирование: интеграционные аспекты,» *Проблемы программирования,* № 2, pp. 280-285, 2000.

[22] И. В. Редько, «Дескриптологическая среда моделирования предметных областей,» *Проблемы программирования,* № 1, pp. 44-50, 2002.

[23] И. В. Редько, «Дескриптивные среды: интеграционные основания,» *Проблеми програмування,* № 1, pp. 17-23, 2006.

[24] И. В. Редько, «Интенсиональные основания дескриптивных сред,» *Проблемы программирования,* № 3, pp. 81-85, 2006.

[25] Д. Робинсон и Э. Зиберт, «Логлисп: основные возможности и реализация,» в *Сборник статей "Логическое программирование"*, Москва, 1988, pp. 261-275.

[26] D. Knuth, "Structured Programming with go to Statements," *Computing Surveys,* vol. 6, no. 4, p. 292, 1974.

[27] В. Н. Агафонов, «Типы и абстракция данных в языках программирования (обзор),» в *Сборник статей "Данные в языках"*, Москва, 1982, pp. 265-327.

[28] Д. Б. Буй и В. Н. Редько, «Неподвижные точки и операторы замыкания: программологические аспекты,» *Кибернетика и системный анализ,* № 1, pp. 113-121, 1995.

[29] Д. Б. Буй и В. Н. Редько, «Программологические аспекты метода неподвижной точки,» *Кибернетика и систем. анализ,* № 5, pp. 158-167, 1994.

[30] В. Н. Редько, «Композиции программ и композиционное программирование,» *Программирование,* № 5, pp. 3-24, 1978.

[31] В. Н. Редько, «Основания композиционного программирования,» *Программирование,* № 3, pp. 3-13, 1979.

[32] В. Н. Редько, «Дескриптологические основания программирования,» *Кибернетика и системный анализ,* № 1, pp. 3-19, 2002.

[33] В. Н. Редько, Ю. Й. Борона, Д. Б. Буй та С. А. Поляков, Реляційні бази даних: табличні алгебри та SQL-подібні мови, Київ: Академперіодика, 2001, p. 197.

[34] В. Н. Редько и Н. С. Никитченко, «Композиционные аспекты программологии. Часть 1,» *Кибернетика,* № 5, pp. 49-56, 1987.

[35] В. Н. Редько и Н. С. Никитченко, «Композиционные аспекты программологии. Часть 2,» *Кибернетика,* № 1, pp. 28-34, 1988.

[36] И. В. Редько, «Дескриптивные аспекты системного подхода,» *Системні дослідження та інформаційні технології,* № 3, pp. 7-28, 2005.

[37] В. Н. Редько, И. В. Редько и Н. В. Гришко, «Дескриптивные системы: концептуальный базис,» *Проблемы программирования,* № 2-3, pp. 75-80, 2006.

[38] В. Н. Редько и И. В. Редько, «Экзистенциальные основания композиционной парадигмы,» *Кибернетика и системный анализ,* № 2, pp. 3-12, 2008.

[39] А. Френкель и И. Бар-Хиллел, Основания теории множеств, Москва: Мир, 1966, p. 326.

[40] Ф. Хаусдорф, Теория множеств, Ленинград: ОНТИ, 1937, p. 304.

[41] И. В. Редько, «Экзистенциальный базис дескриптивных сред,» *Проблемы программирования,* № 1-2, pp. 15-24, 2008.

[42] И. В. Редько, «Экзистенциальный базис сущностных сред,» *Системні дослідження та інформаційні технології,* № 3, pp. 16-31, 2008.

[43] М. Хайдеггер, Бытие и время, С-П.: Санкт-Петербургская издательская фирма «Наука» РАН, 2006, p. 451.

[44] Я. Хинтикка, Логико-эпистемологические исследования, Москва: Прогресс, 1980, p. 448.

[45] Э. Гуссерль, Логические исследования. Картезианские размышления, Минск: Харвест, 2000, p. 752.

[46] Р. Карнап, Значение и необходимость, Москва: Изд. проект «Тривиум», 1958, p. 380.

[47] К. Поппер, Объективное знание. Эволюционный подход, Москва: Эдиториал УРСС, 2002, p. 384.

[48] Г. Фреге, Логика и логическая семантика, Москва: Аспект пресс, 2000, p. 512.

[49] С. К. Клини, Введение в метаматематику, Москва: Издательство иностранной литературы, 1957, p. 526.

[50] Д. П. Горский, Определение, Москва: Мысль, 1974, p. 312.

[51] В. А. Успенский и А. Л. Семенов, Теория алгоритмов: основные открытия и приложения, Москва: Наука, 1987, p. 288.

[52] Ю. А. Шиханович, Введение в современную математику, Москва: Наука, 1965, p. 376.

[53] Э. Дейкстра, Дисциплина программирования, Москва: Мир, 1978, p. 275.

[54] E. W. Dijkstra, "Notes on Structured Programming," in *Structured Programming*, London, Academic Press Ltd., 1972, pp. 1-82.

[55] E. W. Dijkstra, "The End of Computing Science?," *Communications of the ACM,* vol. 44, no. 3, p. 92, 2001.

[56] Н. Вирт, Алгоритмы и структуры данных, Москва: Мир, 1989.

[57] C. A. R. Hoare, "An Axiomatic Bases for Computer Programming," *Communications of the ACM,* vol. 12, no. 10, pp. 576-580?583, 1969.

[58] C. A. R. Hoare and H. Jifeng, Unifying Theories of Programming, London: Prentice Hall Europe, 1998, p. 298.

[59] Г. Буч, Объектно-ориентированный анализ и проектирование, Москва: Бином, 1998, p. 560.

[60] M. Jackson, Software requirements & Specifications, Cambridge: Addison-Wesley, 1995, p. 228.

[61] Д. Б. Буй и И. В. Редько, «Примитивные программные алгебры вычислимых функций,» *Кибернетика,* № 3, pp. 68-74, 1987.

[62] Г. Вригт, Логико-философские исследования, Москва: Прогресс, 1986, p. 595.

[63] Е. К. Войшвилло, Понятие как форма мышления, Москва: Изд-во Московського университета, 1989, p. 239.

[64] Ч. С. Пирс, Принципы философии. Том 2, Санкт-Петербург: Санкт-Петербургское философское общество, 2001, p. 320.

[65] П. Хендерсон, Функциональное программирование, Москва: Мир, 1983, p. 349.

[66] Х. Барендрегт, Ламбда-исчисление, Москва: Мир, 1985.

[67] А. И. Мальцев, Алгоритмы и рекурсивные функции, Москва: Наука, 1965, p. 391.

[68] И. В. Редько, «Открыто-замкнутые основания сред интеграции. Часть 1,» *Системні дослідження та інформаційні технології,* № 4, pp. 7-17, 2010.

[69] D. D. Gajski, F. Vahid, S. Narayan and J. Gong, "SpecSyn: an environment supporting the specify-explore-refine paradigm for hardware/software system design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 6, no. 1, pp. 84-100, 1998.

[70] C. Erbas, A. D. Pimentel and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers,* vol. 55, no. 2, pp. 99-112, 2006.

[71] E. A. DeKock, V. Zivkovic, P. VanDerWolf and E. Deprettere, "Design space exploration of streaming muiliprocessor architectures," in *Proc. of SiPS*, San Diego, 2002.

[72] S. Mohanty and V. Prasanna, "Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures," in *Proceedings of ASIC/SoC Conference, 15th Annual IEEE International*, Rochester, 2002.

[73] R. Zurawsky, Embedded systems handbook, CRC press, 2005, p. 1089.

[74] G. Smith, "Platform Based Design: Does it Answer the Entire SoC Challenge?," in *Proc. of DAC (San Diego, June 7-11, 2004)*, New York, 2004.

[75] S. Leibson, Designing SOCs with configured cores : unleashing the Tensilica Xtensa and diamond cores, Morgan Kaufmann Publishers, 2006, p. 321.

[76] A. DeHon, "The Density Advantage of Configurable Computing," *IEEE Computer,* vol. 33, no. 4, pp. 41-49, 2000.

[77] L. Perre, J. Craninckx and A. Dejonghe, Green Software Defined Radios: Enabling Seamless Connectivity While Saving on Hardware and Energy (1st ed.), Springer, 2008, p. 160.

[78] "ASIC Design, Custom IP & IC Manufacturing," 17 11 2017. [Online]. Available: http://www.esilicon.com/.

[79] J. M. Arnold, "S5: the architecture and development flow of a software configurable processor," in *Field-Programmable Technology, Proceedings International Conference on*, Singapore, 2005.

[80] D. Morris, "Actel Activates Platforms Roadmap Solidifies Embedded Processor Strategy," 2007. [Online]. Available: http://www.eejournal.com/article/20070130-actel/. [Accessed 17 11 2017].

[81] S. Swan, "SystemC transaction level models and RTL verification," in *Design Automation Conference, 43rd, Proceedings of*, San Francisco, 2006.

[82] G. Borriello and K. Hines, "Dynamic communication models in embedded system co-simulation," in *Proceedings of the 34th annual Design Automation Conference*, New York, 1997.

[83] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, New York, 2003.

[84] F. Doucet, R. K. Shyamasundar and Krüger, "Reactivity in SystemC Transaction-Level Models," in *HVC'07 Proceedings of the 3rd international Haifa verification conference on Hardware and software: verification and testing*, Haifa, 2007.

[85] K. Compton, P. Garcia and M. Schulte, "An overview of reconfigurable hardware in embedded systems," in *Proceedings of EURASIP J. Embedded Syst*, 2006.

[86] I. Kuon, R. Tessier and J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135-253, 2007.

[87] J. Adams, A. DeHon, M. deLorimier and N. Kapre, "Design patterns for reconfigurable computing," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004.

[88] C. Jones, J. Villasenor and B. Schoner, "Video Communications using Rapidly Reconfigurable Hardware," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, pp. 565-567, December 1995.

[89] K. N. Chia, J. Villasenor, B. Schoner and C. Zapata, "Configurable Computer Solutions for Automatic Target Recognition," in *Proceedings of FCCM'96*, 1996.

[90] Д. Б. Буй, «Примитивные программные алгебры,» Киев, 1984.

[91] В. М. Глушков, Г. Е. Цейтлин и Е. Л. Ющенко, Алгебра. Языки. Программирование, Київ: Наукова думка, 1974, p. 328.

[92] Г. Е. Цейтлин, Введение в алгоритмику, Київ: Сфера, 1998, p. 310.

[93] Ю. И. Янов, «О логических схемах алгоритмов,» *Проблемы кибернетики*, № 1, pp. 75-127, 1958.

[94] К. Д. Дейт, Введение в системы баз данных, Москва: Мир, 1983, p. 506.

[95] D. Bjorner and C. Jones, Formal Specification and Software Development, London: Printice-Hall Int., 1982, p. 501.

[96] N. Dershowitz and J. P. Jouannaud, "Rewrite Systems," in *Handbook of Theoretical Computer Science*, vol. B: Formal models and semantics, J. v. Leeuwen, Ed., Cambridge, Massachusetts: The MIT Press, 1990, pp. 243-320.

[97] J. R. Abrial, E. Borger and H. Langmaack, Eds., Formal Methods for Industrial Applications. Lecture Notes in Computer Science, London: Springer-Verlag, 1996, p. 506.

[98] В. Н. Редько, «Экспликативное программирование: ретроспективы и перспективы,» в *Труды I Международной научно-практической конференции по программированию*, Киев, 1998.

[99] G. Frege, «Über Sinn und Bedeutung,» *Zeitschrift für Philosophie und philosophische Kritik,* № 100, pp. 25-50, 1982.

[100] В. Н. Редько, «Семантические структуры программ,» *Программирование,* № 1, pp. 3-19, 1981.

[101] В. М. Глушков, «Теория автоматов и формальные преобразования микропрограмм,» *Кибернетика,* № 5, pp. 3-10, 1965.

[102] Б. В. Губский, Е. В. Крапива и И. В. Редько, «Вычислимые функции на реляциях и таблицах в счетном и конечном алфавитах,» *Доклады АН УССР,* № 10, pp. 74-76, 1988.

[103] Д. Б. Буй и И. В. Редько, «Примитивные программные алгебры функций, сохраняющих денотаты,» *Доклады АН УССР,* № 9, pp. 66-68, 1988.

[104] А. И. Мальцев, «Конструктивные алгебры. 1,» *Успехи математических наук,* т. 16, № 3, pp. 3-60, 1961.

[105] Ю. О. Богатирьова, «Обчислюваність на скінченних множинах та мультимножинах,» *Вісник Київського національного університету імені Тараса Шевченка. Серія: фіз.-мат. науки,* № 4, p. 88–96, 2010.

[106] Ю. О. Богатирьова, «Поняття мультимножини. Структура сімейства мультимножин,» в *Матеріали Тринадцятої Міжнародної наукової конференції ім. акад. М. Кравчука*, Київ, 2010.

[107] Д. Б. Буй и И. В. Редько, «Проблемы полноты в классах вычислимых функций, сохраняющих денотаты,» *Программирование,* № 4, pp. 56-68, 1991.

ДОДАТКИ

ДОДАТОК А. АКТИ ВПРОВАДЖЕННЯ

А К Т

про використання наукових та практичних результатів
аспіранта 2015 року випуску кафедри конструювання електронно-обчислювальної апаратури Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського»
Захарченка Тараса Леонідовича,
що викладені в його дисертаційній роботі на здобуття наукового ступеня кандидата технічних наук

Керівництво підприємства ТОВ «Відео Інтернет Технології» (м. Київ) даним актом засвідчує, що запропонована в рамках дисертаційної роботи Захарченка Т.Л. дослідна реалізація адаптивного середовища проектування, призначена для розробки програмного забезпечення обробки зображень використана з метою вдосконалення системи відеоспостереження за рухомими транспортними засобами «Overseer Traffic», що виготовляється даним підприємством.

Використання зазначених результатів дисертації дозволило вирішити проблему верифікації вказаного вище програмного забезпечення та підтримати його розповсюдження на різні обчислювальні платформи, зокрема, на FPGA.

Директор ТОВ
«Відео Інтернет Технології»
04.02.2016

М.П.

Ю.В. Бухтіяров

ЗАТВЕРДЖУЮ
Проректор з наукової роботи Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", акад. НАН України

_____ М.Ю. Ільченко
"2" 12 2016р.

АКТ

використання результатів дисертаційної роботи на здобуття наукового ступеня кандидата технічних наук аспіранта 2015 року випуску Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського" (КПІ ім. Ігоря Сікорського) Захарченка Тараса Леонідовича в НДР

Комісія у складі:

Голови комісії: завідувача кафедри КЕОА КПІ ім. Ігоря Сікорського, д.т.н., проф. О.М. Лисенка;

Членів комісії: в.о. директора НДІ прикладної електроніки КПІ ім. Ігоря Сікорського Д.Д. Татарчука;
професора кафедри КЕОА КПІ ім. Ігоря Сікорського, д.ф-м.н. І.В. Редька

розглянула наукові результати досліджень дисертаційної роботи Захарченка Т.Л. та прийняла рішення про використання розробленої ним дослідної реалізації адаптивного середовища проектування для побудови на її базі надійних та ефективних засобів проектування апаратного забезпечення в НДР "Прискорення обчислень з використанням логічних пристроїв, що реконфігуруються" (РК №0113U001874).

Голова комісії: _____ О.М. Лисенко

Члени комісії: _____ Д.Д. Татарчук

_____ І.В. Редько

ЗАТВЕРДЖУЮ
Перший проректор Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", акад. НАН України

_____ Ю.І. Якименко

"__" _____ 2016р.

АКТ

впровадження результатів дисертаційної роботи аспіранта 2015 року випуску кафедри конструювання електронно-обчислювальної апаратури (КЕОА) Захарченка Тараса Леонідовича в навчальний процес Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського" (КПІ ім. Ігоря Сікорського)

Комісія у складі:

Голови комісії: завідувача кафедри КЕОА КПІ ім. Ігоря Сікорського, д.т.н., проф. О.М. Лисенка;

Членів комісії: професора кафедри КЕОА КПІ ім. Ігоря Сікорського, д.ф-м.н., проф. І.В. Редька;
доцента кафедри КЕОА КПІ ім. Ігоря Сікорського, к.т.н., доц. П.В. Кучернюка

розглянула результати дисертаційної роботи Захарченка Т.Л. та прийняла рішення щодо впровадження їх у навчальний процес кафедри КЕОА КПІ ім. Ігоря Сікорського при підготовці магістрів за спеціальністю "Радіоелектронні апарати та засоби" в дисциплінах "Основи побудови інформаційно-обчислювальних засобів інтеграції" та "Експертні системи" при виконанні лабораторних робіт та в рамках самостійної роботи студентів.

Впровадження результатів дисертаційної роботи Захарченка Т.Л. у навчальний процес дає можливість використовувати магістрантам кафедри КЕОА під час їх навчання та виконання наукових досліджень новітні теоретичні та практичні напрацювання, зокрема, адаптивне середовище проектування, в області розроблення апаратних та програмних засобів з урахуванням різних прагматичних вимог до кінцевого продукту. Як наслідок, це дозволяє підвищити рівень підготовки фахівців на кафедрі.

Голова комісії: _____ О.М. Лисенко

Члени комісії: _____ І.В. Редько

_____ П.В. Кучернюк

ДОДАТОК Б. СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

[1] T. L. Zakharchenko, "Adaptive hardware design," *Технологический аудит и резервы производства,* no. 5, pp. 23-31, 2015.

[2] T. Zakharchenko, "Pragmatics dependent hardware design," in *Proceedings of the 2014 IEEE 34th International Scientific Conference "Electronics and nanotechnology"*, Kyiv, 2014.

[3] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» *Вісник КНУТД,* № 1, pp. 35-39, 2014.

[4] Т. Л. Захарченко, «Исследование возможности реализации эффективной вычислительной архитектуры на реконфигурируемых логических устройствах,» в *Збірник статей VI Міжнародної науково-технічної конференції молодих вчених «Електроніка-2013»*, Київ, 2013.

[5] Т. Л. Захарченко, «Вирішення проблеми повноти в композиційному програмуванні,» в *Матеріали XIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики»*, Київ, 2015.

[6] Т. Л. Захарченко, «Швидкодіючий багатошаровий персептрон, що гнучко масштабується». Україна Патент 72313, 10 08 2012.

[7] Т. Л. Захарченко, І. В. Редько, Д. І. Редько та П. О. Яганов, «Примітивна програмна алгебра обчислюваних функцій над записами,» *Наукові вісті НТУУ «КПІ»,* № 2, pp. 29-40, 2015.

[8] D. I. Redko, I. V. Redko, P. O. Yahanov and T. L. Zakharchenko, "Compositional basis in programmer activity," *Системні дослідження та інформаційні технології,* no. 4, pp. 83-96, 2015.

[9] Т. Л. Захарченко, «Проблема повноти в класі функцій над записами, які зберігають денотати,» *Наукові вісті НТУУ «КПІ»,* № 5, pp. 23-30, 2015.

[10] І. В. Редько, Д. І. Редько та Т. Л. Захарченко, Концептологічні основи проектування, Київ: Компринт, 2016.

ДОДАТОК В. ВІДОМОСТІ ПРО АПРОБАЦІЮ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ

Наукові та практичні результати дисертаційної роботи доповідались і обговорювались на міжнародних та вітчизняних науково-технічних семінарах і конференціях:

- Всеукраїнській науково–практичній конференції «Innovations in science and technology 2012»;
- IEEE 34th International Conference on Electronics and Nanotechnology (ELNANO 2014);
- Міжнародній конференції молодих вчених «Електроніка-2012»;
- Міжнародній конференції молодих вчених «Електроніка-2013»;
- XIII Всеукраїнській науково-практичній конференції студентів, аспірантів і молодих вчених «Теоретичні та прикладні проблеми фізики, інформатики та математики».