Логический язык программирования как инструмент спецификации и верификации динамической памяти

Рене Хаберланд

«СПбГЭТУ», каф. МОЭВМ

21 декабря 2018 г.

научный руководитель: к.т.н., доцент Кринкин, К.В.
к.т.н., доцент Ивановский, С.А.†

Цель работы

Повышение степени выразимости и автоматизации верификации динамической памяти за счёт исключения многозначности между языками спецификации и верификации.

1. Исследование существующих ограничений выразимости и выявление причин расходимости между языками спецификации и верификации.
2. Исследование методов декларативности в утверждениях, которые основываются на предикатной логике. Сравнение реализаций предикатов.
3. Исследование метода верификации динамической памяти на основе выбранного логического языка программирования.
4. Исследование существующих определений «*куч*». Исследование представления куч в унифицированном виде.
5. Разработка и исследование эффективности методов представления куч с помощью программной реализации.
6. Возможность отделения выражений куч от остальных логических правил.
7. Исследование возможности автоматического анализа динамической памяти.

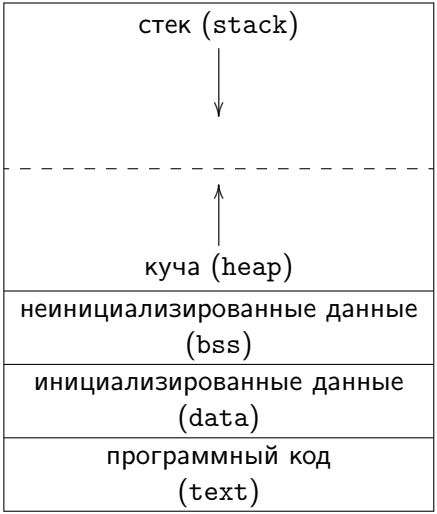
Тройка Хоара

Определение тройки Хоара

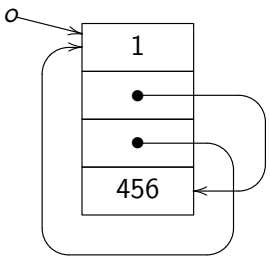
Тройка Хоара состоит из предусловия P до и постусловия Q после выполнения программного оператора C, сокращенно обозначено как $\{P\}C\{Q\}$.

Пример: $(;)\dfrac{\{P\}A_1\{Q\} \quad \{Q\}A_2\{R\}}{\{P\}A_1; A_2\{R\}}$ $\qquad$ $(\Rightarrow)\dfrac{P' \Rightarrow P \quad \{P\}C\{Q\} \quad Q \Rightarrow Q'}{\{P'\}C\{Q'\}}$

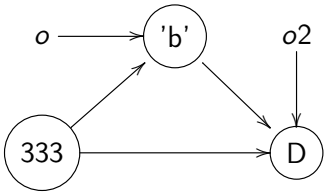
Модель памяти процесса в ОС



Стек
(автоматически)



Heap
(вручную)



Проблемы при работе с динамической памятью

Общие проблемы

1. некорректный доступ к памяти
2. утечка памяти: нехватка памяти + снижение быстродействия

Причины

- Попытки обращения к неверно инициализированной памяти
- Использование указателей вдали от места выделения памяти и инициализации

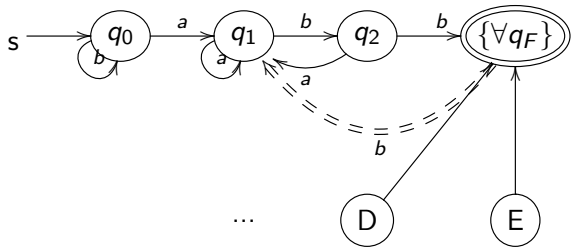
Структура диссертационной работы

- Различие языков спецификации и верификации:
  1. Выявление причин сложности анализа (§4)
  2. Логический язык как язык спецификации и верификации памяти (§3-§6)
- Проблемы с выразимостью описания динамической памяти:
  1. Выявление многозначности в существующих определениях «*кучи*» (§3)
  2. Решение многозначности пространственных операторов (§5)
  3. Решение существующих ограничений спецификации с помощью логического языка (§4)
  4. Включение указателей на ранней стадии разработки программ в UML/OCL (§5)
- Проблемы с автоматизацией:
  1. Расширение и разнообразие промежуточного представления heap (§4)
  2. Алгоритм для автоматического сравнения куч и правил верификации динамической памяти (полнота) (§5,§6)
  3. Универсальный подход к верификации как процессу распознавания базисных куч (§6)

Выразимость формул heap

Проблема адекватного представления

Регулярное выражение или грамматическое представление: $b^*a^+b((a^+b)^* + bba^*b(a^+b)^*)^*b$. Удалим грань a:



$$
\begin{aligned}
Q_0 &= aQ_q + bQ_0 \\
Q_1 &= aQ_1 + bQ_2 \\
Q_2 &= aQ_1 + bQ_F \\
Q_F &= \varepsilon + \underline{bQ_1}
\end{aligned}
$$

(грань b добавляется)

$\Rightarrow$ Локальность наиболее важна; граф представляется компонентами по граням.

Формулы предиката heap

Определение Абстрактного Предиката

$$\Phi ::= true \mid false \mid x \mid REL(f_j(\vec{x})) \mid Pred(f_j(\vec{x})) \mid \neg\Phi \mid \Phi \star \Phi \mid \forall x.\Phi[x] \mid \exists x.\Phi[x]$$

Свойства пространственной разделимости (ЛРП по Рейнольдсу/Бёрдайн)

(1)	несжимаемость:	$p \not\Rightarrow p \star p$, $p \star q \not\Rightarrow p$, если $\exists q, q \not\equiv \mathtt{emp}$
(2)	коммутативность:	$p_1 \star p_2 \Leftrightarrow p_2 \star p_1$
(3)	ассоциативность:	$(p_1 \star p_2) \star p_3 \Leftrightarrow p_1 \star (p_2 \star p_3)$
(4)	нейтральный элемент:	$p \star \mathtt{emp} \Leftrightarrow \mathtt{emp} \star p \Leftrightarrow p$
(5)	дистрибутивность:	$(p_1 \vee p_2) \star q \Leftrightarrow (p_1 \star q) \vee (p_2 \star q)$
		$(p_1 \wedge p_2) \star q \Leftrightarrow (p_1 \star q) \wedge (p_2 \star q)$
(6)	квантификация:	$(\exists x.p) \star q \Leftrightarrow \exists x.(p \star q)$, если $x \notin FV(q)$
		$(\forall x.p) \star q \Leftrightarrow \forall x.(p \star q)$, если $x \notin FV(q)$

Многозначность heap

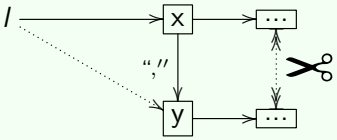
Вопрос

Что такое «*куча*» (не «*кучи*» и не «*подкуча*») ?

Разделимость (оператор $\star$) куч может быть нарушено, когда в разных кучах используются одни и те же термы или символы.

Пример

$$tree(l) ::= \texttt{nil} \mid \exists x.\exists y : \; l \mapsto x, y \; \star \; tree(x) \; \star \; tree(y)$$



Вывод: $\star$ не (только) делит, $\mapsto$ определяет базисную кучу.

Преобразование в Пролог

Определение терма

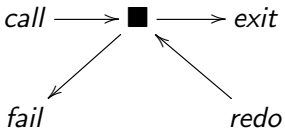
$$
T ::= \begin{cases}
x & \text{символ } x \in \overline{X} \text{ из множества допустимых символов} \\
X & \text{символьная переменная } X \in \overline{X} \\
[] & \text{пустой список} \\
[\ T \mid Ts\ ] & \text{терм голова списка } T \in \overline{X}, \text{ где } Ts \text{ остаток списка} \\
[\ T_0\ ,\ \ldots\ ,\ T_n] & \text{список с } T_j \text{ терм, } 0 < j \leq n \\
f(T_0, \ldots, T_n) & f \text{ функтор, } T_j \text{ термы, } 0 \leq j \leq n \\
p(T_0, \ldots, T_n) & p \text{ предикат, } T_j \text{ термы, } 0 \leq j \leq n
\end{cases}
$$

- При верификации производится сравнение (унификация) между термом спецификации и актуальным термом, соотношения имеют особое значение.
- Преобразование термов куч производится согласно данным формальным теориям куч.
- Правила верификации heap записываются, как правила Хорна.

Логическое программирование как метод доказательства

Преобразование в задачу логического вывода

- Подключение (логического) программирования как наиболее *гибкая* и выразительная схема.

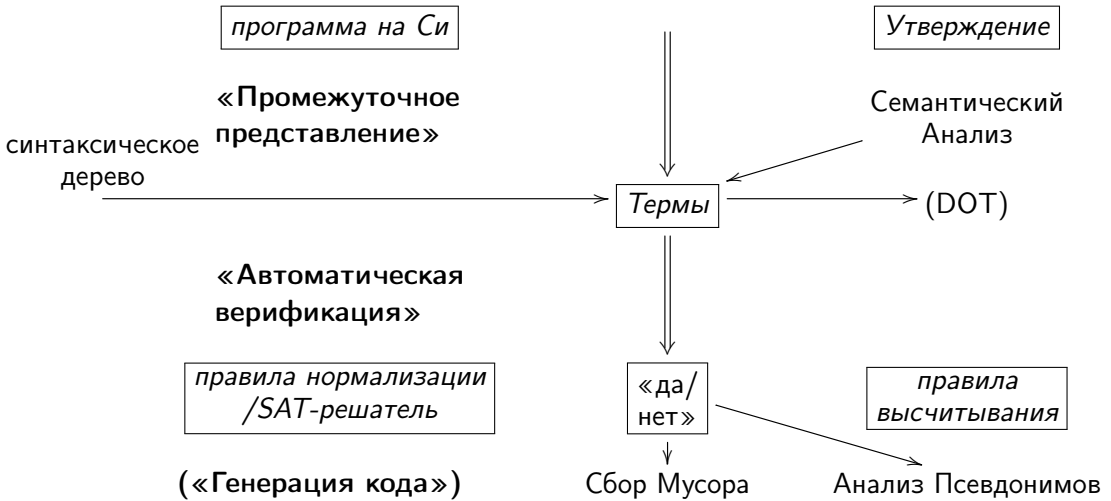


(2-ая семантика: процедура + предикат) соответствует порядку верификации в вычислении Хоара, включая стратегию *backtracking*, раскручивание и окна стека

- Центральная проблема: расположение/соотношение — это логическая проблема
- Декларативность (описание графа heap) способствует кэшированию подцелей
- Функторы вводят λ-термы 3-его порядка (объекты)

Логическое программирование:	
Входные данные:	правила Пролога, запрос
Выходные данные:	ответ «*Да/Нет*», дерево вывода и все сопоставления

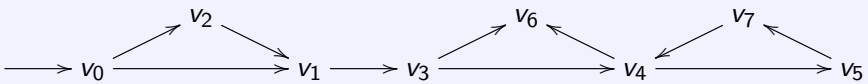
Архитектура верификатора динамической памяти



Ужесточение выразимости языка спецификации куч

Проблемы разделения сложных куч

(№1) Разделение на абстрактные предикаты



Как лучше делить: по грани $v_1 \to v_3$ или по вершине v_4 и т.д. ?

(№2) Многозначность распределённого оператора

Разделение «$\star$» не делит строго, т.к. из аксиомы №6 ЛРП:

(6) квантификация: $(\exists x.p) \star q \Leftrightarrow \exists x.(p \star q)$, если $x \notin FV(q)$

$(\forall x.p) \star q \Leftrightarrow \forall x.(p \star q)$, если $x \notin FV(q)$

$\Rightarrow$ общая вершина графа кучи не исключена при $\star$ (ср. Smallfoot, SpaceInvader и т.д.).

Ослабить требование (разбить неявную «*делимость*» на явное разделение и явное слияние) — это означает ужесточить разделимость.

Многозначность пространственного оператора

Минимальный пример-аналогия

`Expression ::= value | Expression` $\otimes$ `Expression`

Если имеются зависимости между подвыражениями, то анализ включает в себя весь контекст. Другими словами, оператор $\otimes$ перегружен.

Требование адекватности

Простая куча должна описываться простой моделью.

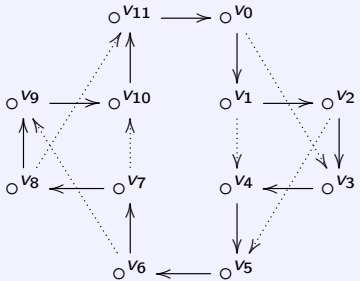
Требование корректности

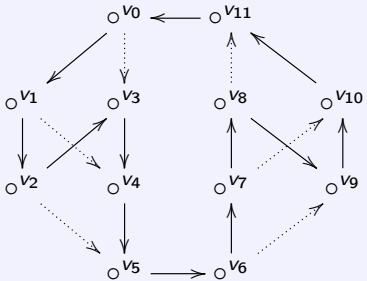
Только синтаксически верные кучи соответствуют модели кучи. Некорректные кучи не должны логически выводиться из описания.

Проблема преобразования куч

(№3) Изоморфизмы смежных куч с объектами

- При сравнении куч проблема решена (из-за локации).





Выводимое свойство (неужесточённое) разделимости по графу

Дано

- куча $H_1 = (V_1, E_1)$, куча $H_2 = (V_2, E_2)$

$H = H_1 \star H_2$ (по Рейнольдсу)

Результат: куча $H = (V, E)$ где:

1. $E = V \times V$
2. $\forall v_1 \in V_1,\ v_2 \in V_2$
3. $v1 \neq v2$ и $V_1, V_2 \subseteq V$
4. (*Разделение*): $(v_1, v_2) \notin E_1$, и $(v_1, v_2) \notin E_2$.
5. (*Слияние*): $\exists s \in V_1, \exists t \in V_2 : (s, t) \in E_1$ или $(s, t) \in E_2$, тогда H_1 или H_2 содержит $\star$-разделённые $s \mapsto t$.

Строгая конъюнкция графов кучи

Дано

куча $H = (V, E)$, базисная куча $\alpha \mapsto \beta$

$H' = (V, E) \circ \alpha \mapsto \beta$

$$\begin{cases} (V \cup \{\alpha, \beta\} \cup \beta', & \text{если } \mathit{isFreeIn}(\alpha, H) \\ E \cup \{(\alpha, \beta)\} \cup \{(\beta, b) | b \in \beta'\}) & \text{если } H = \underline{emp} \\ & \quad (V = E = \emptyset) \\ \underline{\text{false}} & \text{иначе} \end{cases}$$

Конвенция

- $H \circ \underline{emp} \equiv \underline{emp} \circ H \equiv H$
- Оператор доступа к объектному полю «.» лево-ассоциативен.

Результаты

Результаты (I)

(1—3) $\models$ Унифицированный язык спецификации и верификации куч

(1—4) $\models$ Принципы архитектуры верификатора куч

(4) $\models$ Повышение выразимости формул куч (например символы, предикаты)

(4) $\models$ Реляционная модель для описания куч

(5) $\models$ Прототип архитектуры верификатора куч

Результаты (II)

(4,6) $\models$ Интерпретация и введение новых формальных теорий куч

(4,7) $\models$ Новое определение единственной кучи в виде графа

(6,7) $\models$ Упрощение спецификации куч с помощью введённых $\circ$ и $||$

(6) $\models$ Предложено расширение объектного вычисления с указателями

(7) $\models$ Доказано, что абстрактные предикаты в Пролог-диалекте представляют атрибутируемую кучу

Апробация работы

Основные результаты работы докладывались научных конференциях докладывались и обсуждались на следующих конференциях:

1. The 10th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP), October 9-13 2016, Venice, Italy.
2. 18th Conference of Open Innovations (FRUCT), April 18-22 2016, St-Petersburg, Russia.
3. EMC^2 «Технологии Майкрософт в Теории и на Практике Программирования: Новые подходы к разработке программного обеспечения», September 9th 2014, St-Petersburg, Russia
4. 8th Spring/Summer Young Researcher's Colloquium on Software Engineering (SYRCoSE), May 29-31 2014, St-Petersburg 2014.
5. 10th Conference on Advances in Methods of Information and Communication Technology (AMICT), May 20-21 2008, Petrozavodsk, Russia/Helsinki, Finland.
6. Intl. Scientific Symposium Sense Enable (SPITSE 2016), Russian National Research University, June 20-24 2016, Moscow, Russia.
7. Intl. Conference on Control Processes and Stability, April 4-7 2008, St-Petersburg, Russia.

Опубликованные статьи — А

- Хаберланд, Р. Верификация Объектно-ориентированных программ с динамической памятью на основе ссылочной модели с помощью Пролога. / Р. Хаберланд // Известия ЛЭТИ, Санкт Петербургский Электротехнический Университет. – №1. – 2016. — Санкт Петербург. — С.14-17.
- René Haberland, Kirill Krinkin. *A Non-repetitive Logic for Verification of Dynamic Memory with Explicit Heap Conjunction and Disjunction*. In: International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP), ISBN 978-1-61208-506-7. Ed. University of Venice, Italy. Oct. 2016, pp. 1–9.
- René Haberland, Kirill Krinkin, Sergey Ivanovskiy. *Abstract Predicate Entailment over Points-To Heaplets is Syntax Recognition*. In: 18th Conference of Open Innovations (FRUCT), ISSN 2305-7254, ISBN 978-952-68397-3-8. Ed. By T. Tyutina S. Balandin A. Levina. 2016, pp. 66–74.

B

- Хаберланд, Р. Верификация корректности динамической памяти с помощью логического языка программирования. / Р. Хаберланд // Конференция EMC^2 «Технологии Майкрософт в Теории и на Практике Программирования. Серия: Новые подходы к разработке программного обеспечения», С.-П. Политехнический Университет. – №3. – 2014. – Санкт Петербург. – С.56-57.
- René Haberland, Sergey Ivanovskiy. *Dynamically Allocated Memory Verification in Object-Oriented Programs using Prolog*. In: Proc. of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2014). Ed. by Alexander Kamkin, Alexander Petrenko, and Andrey Terekhov. 2014, pp. 46–50.
- René Haberland,Igor L. Bratchikov. *Transformation of XML Documents with Prolog*. In: Advances in Methods of Information and Communication Technology. Vol. 10. 2008, pp. 99–111.

C

- René Haberland. *A Stricter Heap Separating Points-To Logic*. In: 3rd Int. Scientific Symposium Sense Enable (SPITSE2016). Ed. by Russia National Research University Moscow. June 2016, pp. 14–15.
- Хаберланд, Р. Расширяемый Фреймворк для верификации статически типизированных объектно-ориентированных программ, использующих динамическую память. / Р. Хаберланд // Санкт Петербургский Электротехнический Университет «ЛЭТИ» (ППС ЛЭТИ). – №5 – дек. 2015 – Санкт Петербург.
- René Haberland. *Unification of Template-Expansion and XML-Validation*. In: Proc. of Int. Conf. on Control Processes and Stability. Vol. 34. Saint Petersburg State University, 2008, pp. 389–394.
- Хаберланд, Р., Кринкин К.В. Сравнительный анализ статитических методов верификации динамической памяти — Деп. в ВИНИТИ РАН, 27.07.2018г. №89-В2018.

Дополнительные слайды

Новизна

1. Исследован разрыв между языками спец./вериф. дин. памяти, предложен логический язык программирования как инструмент.
2. Проведены сравнения выразимости при трансформации термов в логическом и функциональном представлениях/императивные программные операторы.
3. Предложено, термы диалекта языка Пролог использовать непосредственно для представления данных полного конвейера статического анализа дин. памяти.
4. Предложен автоматизированный процесс верификации как синтак. перебор абстрактных предикатов.
5. Сняты ограничения выразимости переменных символов, термов и рек. предикатов, а также ограничения всязи с встроенными предикатами.
6. Повышена выразимость за счёт ужесточения операций над кучами.
7. Установлены алгебраические свойства над кучами.
8. Достигнуто повышение выразимости куч и полноты правил, за счёт введения частичного оператора «_».

Практическая значимость

При автоматизированной поддержке вывода с повышенной выразимостью теоретически обоснованный и разработанный прототип на основе диалекта Пролога позволяет проводить верификацию проще и короче.
С практической точки зрения, добавление новых языковых возможностей языка программирования приводит к новым проблемам между спецификацией и верификацией куч. таким образом: (1) необходимо проводить спецификации исключительно на логическом/декларативном языке без побочных эффектов, например близком к Прологу; (2) любое представление на языке спецификации должно обрабатываться элементами языка верификации, какова бы ни была система логического вывода; (3) языки спецификации и верификации имеют сильные пересечения, и поэтому унификация обоих языков упрощает оба процесса.
Разработанные в рамках диссертации решения (платформа для верификации, далее – Платформа) разрешают добавлять новые фазы по обработке динамической памяти, менять существующие и переводить данные Си-программы в термовое представление. В качестве входного языка может быть использован любой другой язык, в том числе

История существующих работ

Floyd67	Assigning Meanings to Programs
Hoare69	An Axiomatic Basis for Computer Programming
Burstall72	Techniques for Programs that Alter Data Structures
Kowalski74	Predicate Logic as Programming Language
Clarke79	Why some programming constructs are hard to formalise?
Apt81	Ten Years of Hoare's Logic
Suzuki82	Analysis of Pointer Rotation
Warren83	Applied Logic – Its Use and Implementation as a Programming Tool
Horwitz89	Dependence Analysis for Pointer Variables
Miller90	An Empirical Study of the Reliability of UNIX Utilities
Steinbach94	Termination of Rewriting
Tofte94	Region Calculus
Abadi96	A Theory of Objects
Hutton98	Fold and Unfold for Program Semantics

История существующих работ

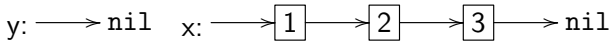
Leino98	Recursive Object Types in a Logic of Object-Oriented Programs
Matthews98	An Introduction to Natural Language Processing Through Prolog
Bornat00	Proving Pointer Programs in Hoare Logic
Reynolds02	Separation Logic: A Logic for Shared Mutable Data Structures
Meyer03	Proving Pointer Program Properties
Bertot04	Coq — The Calculus of Inductive Constructions
Parkinson05	Local Reasoning for Java
Berdine05	Smallfoot (Separation Logic)
Hurlin07	Specification and Verification of Multithreaded OO Programs with SL
Birkedal08	A Simple Model of Separation Logic for HOS
Distefano08	jStar: Towards Practical Verification for Java
Dodds08	Graph Transformations and Pointer Structures
Parduhn08	Algorithm Visualization using Concrete and Abstract Shape Graphs
Calcagno09	Compositional Shape Analysis by means of Bi-abduction
Jacobs11	VeriFast

Пример линейного списка на Си

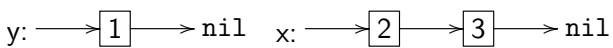
Реверс списка (по Рейнольдсу)

Состояние динамической памяти:

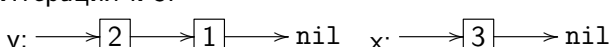
Итерация №1:



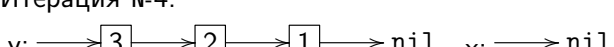
Итерация №2:



Итерация №3:



Итерация №4:



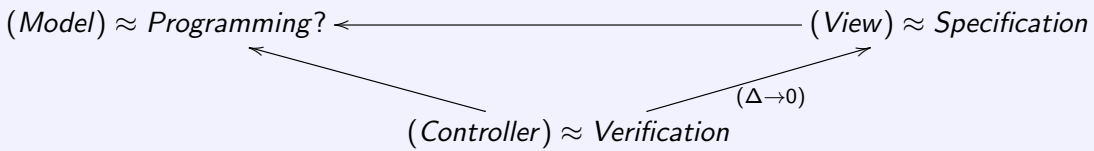
Си:

```
list *temp;
list *y = NULL;
// param list *x
while (x != NULL){
  temp = x->next;
  x->next = y;
  y = x;
  x = temp;
}
```

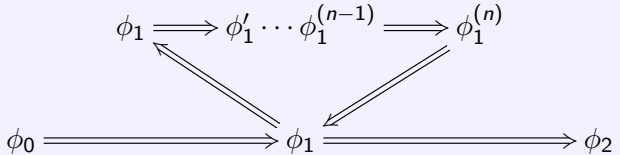
```
int *x;
int *good(){ int *p = x; return p; }
int *bad() { int x=55; return &x; }

void main(){
 int a=5,b=4;     int *pa = &a;
 const int *const pa2 = &a;
 a=77;
 //pa2 = pa; // 'pa2' is read-only
 printf("*pa==%d,␣*pa2==%d\n",*pa,*pa2); // *pa==77, *pa2==77
 a=*&b;  // // *const of pa2 is still no guarantee for safety
 printf("*pa==%d,␣*pa2==%d\n",*pa,*pa2); // ! *pa==4, *pa2==4

 x = (int*)malloc(10); memset(x,7,sizeof(x));
 printf("after␣good:␣0x%x\n", *(good())); // OK
 printf("after␣bad:␣0x%x\n", *(bad())); // SEGV
}
```

Принципы построения логической системы для верификации куч

Роли языков, причастных к верификации куч



Архитектура конвейера верификатора и статических анализаторов



Пример верификация на *Coq*

```
Definition peirce := forall (p q: Prop), ((p->q)->p)->p.
Definition lem := forall p, p \/ ~p.
Theorem peirce_equiv_lem: peirce <-> lem.
```

After unfold: 1 subgoal (1/1)

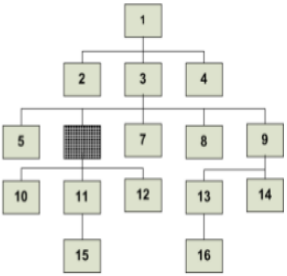
```
(∀ p q: Prop,
((p→q)→p)→p)⇔(∀ p: Prop,
p ∨¬ p)
```

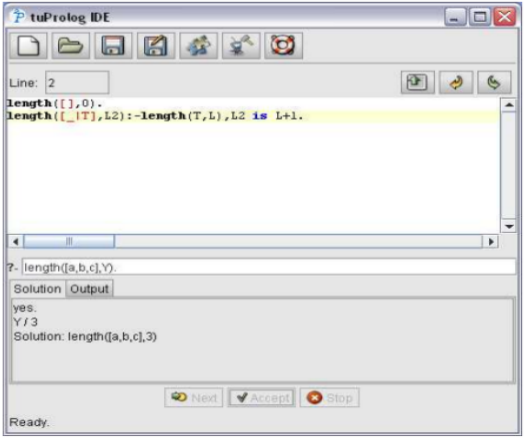
After apply: 2 subgoals

```
H: ∀ p q: Prop, ((p→q)→p)→p
p: Prop                          (1/2)
(p ∨¬ p → ¬(p ∨¬ p)) → p ∨¬ p    (2/2)
p
```

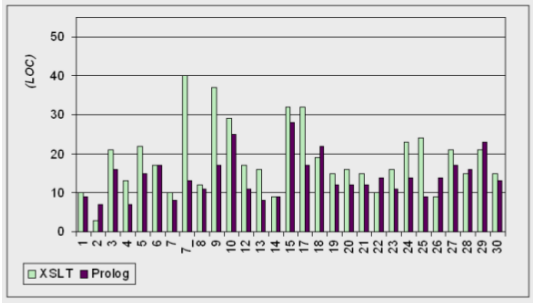
```
Proof.
  unfold peirce, lem.
  firstorder.
  apply H with (q:=~(p \/ ~p)).
  firstorder.
  destruct (H p).
  assumption.
  tauto.
Qed.
```

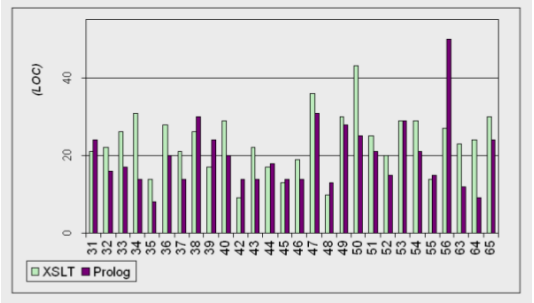
Сравнение функционалов — термы I



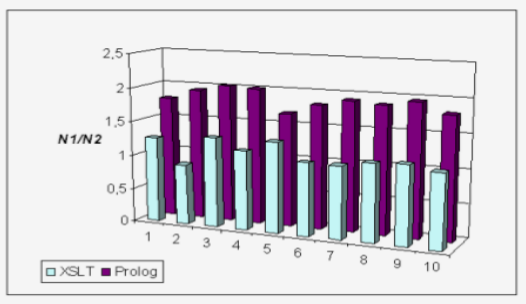


Сравнение функционалов — термы II





Сравнение функционалов — термы III



Выразимость абстрактных предикатов (I)

Объединение реляций:

$T = R \cup S$: $t(x_1, \cdots, x_m) : -r(x_1, \cdots, x_m).$
$t(y_1, \cdots, y_n) : -s(y_1, \cdots, y_n).$

Множественный минус:

$T = R/S$: $t(x_1, \cdots, x_n) : -r(x_1, \cdots, x_n), not\_s(x_1, \cdots, x_n).$

Картезиянский продукт:

$T = R \times S$: $t(x_1, \cdots, x_m, y_1, \cdots, y_n) : -r(x_1, \cdots, x_m), s(y_1, \cdots, y_n).$

Проекция:

$T = \Pi_S(R)$: $t(s_1, \cdots, s_n) : -r(x_1, \cdots, x_m).\ \forall s_i \in \{x_1, \cdots, x_m\}$

Выбор/Селекция:

$T = \sigma_S(R)$: $t(x_1, \cdots, x_n) : -r(x_1, \cdots, x_n), s(x_1, \cdots, x_n).$

Переименование:

$T = \rho_S(R)$: $t(x_1, \cdots, x_n) : -r(x_1, \cdots, x_n).$

$\Rightarrow$ Коддская алгебра реляций соблюдается, а значит априори имеется высокое практическое значение.

Выразимость абстрактных предикатов (II)

Снятые ограничения (ср. Паркинсон, Бёрдайн и другие):

1. Описание символов и термов (в том числе и соотношений) компактнее + более выразительно, если парадигма дескриптивна и логична
2. Термы heap унифицируемы и не нуждаются в (не полных) определениях и всё новых конвенциях
3. Параметр/символ в heap и абстрактных предикатах может находиться везде

Далее снятые ограничения:

1. Абстрактные предикаты (не функционалы) как настоящие предикаты Пролога
2. Перегрузка значений предикатов возможна (включая обратимость)
3. Сравнение термов снаружи-во-внутрь (генерация контр-примеров)
4. Терм heap упрощается с помощью данных теорий
5. Объекты в памяти непрерывны (включая указатели)

Свойства графа-кучи (I)

- оператор $\circ$ тотален
- множество куч Ω перечислимое и возможно бесконечное
- $G = (\Omega, \circ)$ является моноидом и даже (Абелевой) группой
  $\Rightarrow$ вводится «*обратимая куча*» и конвенция: $\forall \omega \in \Omega. \omega \circ \omega^{-1} \equiv \omega^{-1} \circ \omega \equiv \varepsilon \equiv \underline{emp}$.
  $\Rightarrow$ при нейтрализации обобщённых куч необходимо проводить нормализацию
  $\Rightarrow (g_1 \circ g_2)^{-1} \equiv g_1^{-1} \circ g_2^{-1}$

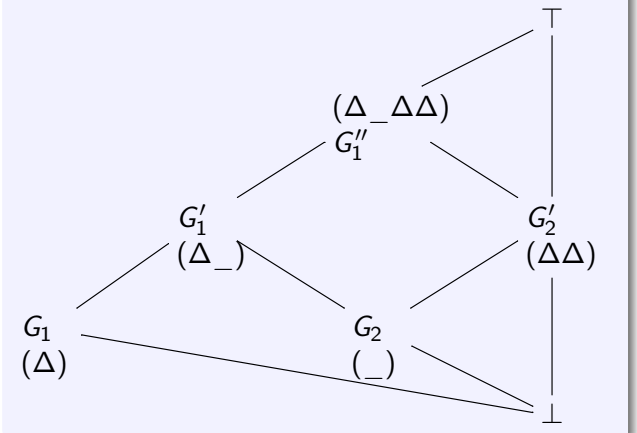
Термовое выражение кучи

$T ::=$	$loc \mapsto val$	... *обыкновенная (базисная) куча*
	$\mid T \circ T$	... *конъюнкция*
	$\mid T \mid\mid T$	... *дизъюнкция (приоритет ниже чем у* $\circ$*)*
	$\mid \underline{true} \mid \underline{false} \mid \underline{emp}$	... *константные предикаты куч*

- $\circ$ и $\mid\mid$ дуальны. Аналогичные свойства в силе для дизъюнкции куч

Свойства графа-кучи (II)

Частично-упорядоченное множество $(H_\circ; \sqsubseteq)$



Дистрибутивность

(i) $a \circ (b \| c) = (a \circ b) \| (a \circ c)$

(ii) $(b \| c) \circ a = (b \circ a) \| (c \circ a)$

- ЧУМ $\Rightarrow$ линейный анализ термов
- (i)+(ii) $\Rightarrow$ НФ всегда существует
- операторы определены «явно» и над «графом»

Частичная спецификация

- Экземпляр содержит связанные поля. Константы к нему также применяются, как и к отдельным указателям
- Пример: Экземпляр a имеет поля f_1, g_1 и g_2, тогда преобразователь $\mathcal{C}[\![\,.\,]\!]$:

$$\begin{aligned} & \mathcal{C}[\![a.f_1 \mapsto x \circ \underline{true}(a)]\!] = \quad \mathcal{C}[\![a.f_1 \circ a.g_1 \circ a.g_2]\!] \\ = \quad & \mathcal{C}[\![\underline{true}(a) \circ a.f_1 \mapsto x]\!] \neq \quad \mathcal{C}[\![p(a) \circ a.f_1 \mapsto x]\!] \end{aligned}$$

- $\circ$ и $||$ позволяют высчитывать и сравнивать термы выражений куч (для теорий и в будущем для абдукции)
- $\Rightarrow$ частичная (не полная) спецификация приводит к проверке и выявлению неполного набора правил.

Автоматизация верификации с предикатами

Произведение базисных куч

- Рассмотрим абстрактные предикаты более подробно:

Пример:

$\exists v1.v2.v3$, при $\texttt{p1.data} \mapsto v1 \star \texttt{p2.data} \mapsto v2 \star \texttt{p3.data} \mapsto v3 \star \texttt{p1.next} \mapsto \texttt{p2} \star \texttt{p2.next} \mapsto \texttt{p3} \star \texttt{p3.next} \mapsto \texttt{p1} \star \texttt{p1.prev} \mapsto \texttt{p3} \star \texttt{p3.prev} \mapsto \texttt{p2} \star \texttt{p2.prev} \mapsto \texttt{p1}$

может быть лучше читаемо и понятно (в связи с ходом доказательства) как:
`face(p1,p2,p3)`.

- Например, Хаттон и Джоунс предлагают использовать «*Fold/Unfold*» как метод доказательства корректности функционалов (ручное доказательство).

Свёртка/Развёртывание

- Интерпретируем базисные кучи как терминалы (с параметрами), сложные — как нетерминалы. Тогда соотношения унификации также разрешаются:

Атрибутируемая грамматика

$face_{X,Y,Z} \to X \mapsto Y \star Y \mapsto Z \star Z \mapsto X$
$polygon1L_{U,V,W,X,Y,Z} \to face_{X,Y,Z} \star U \mapsto X \star face_{V,W,Y}$
$polygon1L_{U,V,W,X,Y,Z} \to face_{X,Z,Y} \star X \mapsto X \star face_{V,W,Y}$
$polygon3R_{W,U,V,\ldots} \to \ldots \star \ldots polygon3R_{\ldots} \star \ldots \star polygon1L_{\ldots}$

- Атрибуты присвоены для конкретных подцелей
- Преобразование проблемы верификации куч в проблему распознавания куч (с помощью π, σ) = логическое программирование

Свёртка/Развёртывание (II)

Синтаксический анализ:

- Входные данные: формальная грамматика, входное слово
  Выходные данные: ответ «*Да/Нет*», дерево вывода

Пример абстрактного предиката

```
face(X,Y,Z):-pointsto(X,Y),pointsto(Y,Z),pointsto(Z,X).
polygon1L(U,V,W,X,Y,Z):-face(X,Y,Z),pointsto(U,X),face(V,W,Y).
polygon1L(U,V,W,X,Y,Z):-face(X,Z,Y),pointsto(X,X),face(V,W,Y).
polygon3R(W,U,V):- ... , polygon3R(...), ..., polygon1L(...) .
```

- $\star$ здесь обозначает только конъюнкцию куч.
- Некоторые предикаты (как pointsto) имеют специальное назначение.
- Дизъюнкция определяется альтернативой в Хорнских правилах («;»).