**Лекция 1. Что изучает логика. Логика в информатике. Структура курса...**

ЛОГИКА — междисциплинарная отрасль наук, изучающая

1. законы причинно-следственной связи в окружающем мире;
2. проявление причинно-следственных законов в рациональном мышлении человека;
3. отражение причинно-следственных законов в языках (естественных и искусственных).

ФОРМАЛЬНАЯ ЛОГИКА – изучает формы, в которых проявляются законы причинно-следственных связей, вне зависимости от содержания (смысла) тех явлений (предметов), к которым эти законы относятся.

Основная задача формальной логики:

База знаний: Γ = {ϕ1, ϕ2, …, ϕN}.

Предложение: ψ.

Задача (неформальная): выяснить, является ли предложение ψ следствием утверждений базы знаний Γ.

Задача (формальная): проверить, что ψ выводится из Γ по законам формальной логики.

**Лекция 2. Классическая логика предикатов первого порядка. Синтаксис. Термы и формулы. Семантика. Интерпретация. Выполнимость формул.**

Базовые символы.

Предметные переменные Var = {x1, x2, . . . , xk, . . . };

Предметные константы Const = {c1, c2, . . . , cl, . . . } – это имена предметов

Функциональные символы Func = {f(n1)1, f(n2)2, . . . , f(nr )r, . . . } – это операции над предметами

Предикатные символы Pred = {P(m1)1, P(m2)2, . . . , P(ms )s , . . . } – это отношения между предметами.

Тройка <Const , Pred , Func> называется сигнатурой алфавита.

Терм — это

1. x , если x ∈ Var (x — переменная)
2. c , если c ∈ Const (c — константа)
3. f(n)(t1, t2, . . . , tn) , если f(n)∈ Func и (t1, t2, . . . , tn — термы) – составной терм.

*Term* — множество термов заданного алфавита.

*Var t*— множество переменных, входящих в состав терма t.

t (x1, x2, . . . , xn) — запись обозначающая терм t , у которого Var t⊆ {x1, x2, . . . , xn}.

Если Vart = ∅, то терм t называется основным термом.

Формула — это атомарная формула P(m)(t1, t2, . . . , tm) , если P(m)∈ Pred , {t1, t2, . . . , tm} ⊆ Term;

составная формула:

(ϕ&ψ), (ϕ ∨ ψ), (ϕ → ψ), (¬ϕ), если ϕ, ψ — формулы;

(∃x ϕ), (∀x ϕ) , если x ∈ Var , ϕ — формула.

Form — множество всех формул заданного алфавита.

Свободные и связанные переменные.

Квантор связывает ту переменную, которая следует за ним. Вхождение переменной в области действия квантора,

связывающего эту переменную, называется связанным. Вхождение переменной в формулу, не являющееся связанным, называется свободным. Переменная называется свободной, если она имеет свободное вхождение в формулу.

Var ϕ — множество свободных переменных формулы ϕ.

ϕ(x1, x2, . . . , xn) — запись, обозначающая формулу ϕ, у которой Var ϕ⊆ {x1, x2, . . . , xn}.

Если Varϕ = ∅, то формула ϕ называется замкнутой формулой, или предложением.

CForm — множество всех замкнутых формул.

Приоритет логических операций:

¬, ∀, ∃ & ∨ →

Семантика — это свод правил, наделяющих значением (смыслом) синтаксические конструкции языка.

Значения термов и формул определяются на основе алгебраических систем.

Алгебраические системы, используемые в таком качестве, называются интерпретациями.

Интерпретация сигнатуры <Const, Func, Pred> — это алгебраическая система I = <DI, Const, Func, Pred>, где

1. DI— непустое множество, которое называется областью интерпретации, предметной областью, или универсумом
2. Const (с чертой сверху): Const → DI — оценка констант, сопоставляющая каждой константе c элемент (предмет) c (с чертой сверху) из области интерпретации;
3. Func: Func(n) → (DnI → DI) — оценка функциональных символов, сопоставляющая каждому функциональному символу f(n) местности n всюду определенную n-местную функцию f(n) на области интерпретации;
4. Pred : Pred(m) → (DmI → {true, false}) — оценка предикатных символов, сопоставляющая каждому предикатному символу P(m) местности m всюду определенное m-местное отношение P(m) на области интерпретации.

Значение терма

Пусть заданы интерпретация I = <DI, Const, Func, Pred>, терм t(x1, x2, . . . , xn) и набор d1, d2, . . . , dn элементов (предметов) из области интерпретации DI.

Значение t(x1, x2, . . . , xn)[d1, d2, . . . , dn] терма t (x1, x2, . . . , xn)на наборе d1, d2, . . . , dn определяется рекурсивно

1. Если t (x1, x2, . . . , xn) = xi, то t(x1, x2, . . . , xn)[d1, d2, . . . , dn] = di
2. Если t (x1, x2, . . . , xn) = c, то t(x1, x2, . . . , xn)[d1, d2, . . . , dn] = c(с чертой сверху)
3. Если t (x1, x2, . . . , xn) = f(t1, . . . , tk), то t(x1, x2, . . . , xn)[d1, d2, . . . , dn] = f(t1[d1, d2, . . . , dn], . . . , tk[d1, d2, . . . , dn]) (с чертой сверху)

Отношение выполнимости формул

Значение формул в интерпретации определяется при помощи отношения выполнимости |=. Пусть заданы интерпретация I = <DI, Const, Func, Pred>, формула ϕ(x1, x2, . . . , xn) и набор d1, d2, . . . , dn элементов (предметов) из области интерпретации DI.

Отношение выполнимости I |= ϕ(x1, x2, . . . , xn)[d1, d2, . . . , dn], формулы ϕ в интерпретации I на наборе d1, d2, . . ., dn определяется рекурсивно – Если ϕ(x1, x2, . . . , xn) = P(t1, . . . , tm), то I |= ϕ(x1, x2, . . . , xn)[d1, d2, . . . , dn] ⇐⇒ P(t1[d1, d2, . . . , dn], . . . , tm[d1, d2, . . . , dn]) (с чертой сверху) = true

**Лекция 3. Выполнимые и общезначимые формулы. Модели. Логическое следование. Проблема общезначимости. Семантические таблицы.**

Формула ϕ(x1, . . . , xn) называется выполнимой в интерпретации I, если существует такой набор элементов d1, . . . , dn ∈ DI, для которого имеет место I |= ϕ(x1, . . . , xn)[d1, . . . , dn].

Формула ϕ(x1, . . . , xn) называется истинной в интерпретации I, если для любого набора элементов d1, . . . , dn ∈ DI имеет место I |= ϕ(x1, . . . , xn)[d1, . . . , dn].

Формула ϕ(x1, . . . , xn) называется выполнимой, если есть интерпретация I, в которой эта формула выполнима.

Формула ϕ(x1, . . . , xn) называется общезначимой (или тождественно истинной), если эта формула истинна в любой интерпретации.

Формула ϕ(x1, . . . , xn) называется противоречивой (или невыполнимой), если она не является выполнимой.

Пусть Γ — некоторое множество замкнутых формул, Γ ⊆ CForm. Тогда каждая интерпретация I, в которой выполняются все формулы множества Γ, называется моделью для множества Γ.

Модель для множества формул Γ — это интерпретация (реальный или виртуальный мир), устройство которого адекватно всем предложениям из множества Γ.

Пусть Γ — некоторое множество замкнутых формул, и ϕ — замкнутая формула. Формула ϕ называется логическим следствием множества предложений (базы знаний) Γ, если каждая модель для множества формул Γ является моделью для формулы ϕ,

т. е. для любой интерпретации I : I |= Γ ⇒ I |= ϕ

Логические следствия — это «производные» знания, которые неизбежно сопутствуют «базовым» знаниям Γ, находятся в причинно-следственной зависимости от предложений Γ. Одна из главных задач (и одновременно наиболее характерное проявление) интеллектуальной деятельности — это извлечение логических следствий из баз знаний.

Запись Γ |= ϕ обозначает, что ϕ — логическое следствие Γ .

Поэтому для обозначения общезначимости формулы ϕ будем использовать запись |= ϕ.

Теорема о логическом следствии

Пусть Γ = {ψ1, . . . , ψn} ⊆ CForm, ϕ ∈ CForm. Тогда Γ |= ϕ ⇐⇒ |= (ψ1& . . . &ψn → ϕ).

(доказательство – по определениям)

Утверждение.

Для любой формулы ϕ(x1, . . . , xn) верно, что

1. |= ϕ(x1, . . . , xn) ⇐⇒ |= ∀x1. . . ∀xn ϕ(x1, . . . , xn);
2. ϕ(x1, . . . , xn) — выполнимая ⇐⇒ ∃x1. . . ∃xn ϕ(x1, . . . , xn) — выполнимая;
3. ϕ(x1, . . . , xn) — выполнима в любой интерпретации ⇐⇒ |= ∃x1. . . ∃xn ϕ(x1, . . . , xn).

Утверждение.

Существует такая замкнутая формула ϕ, которая истинна в любой интерпретации I с конечной предметной областью DI, но не является общезначимой.

(из свойства иррефлексивности и транзитивности -> «свойство максимального элемента», доказательство не строгое – через аналогию с начальником, который командует подчинённым (одно число больше другого))

Семантическая таблица — это упорядоченная пара множеств формул <Γ ; ∆>, Γ, ∆ ⊆ Form.

Γ — это множество формул, которые мы хотим считать истинными, ∆ — это множество формул, которые мы хотим считать ложными.

Пусть {x1, x2, . . . , xn} — множество свободных переменных в формулах множеств Γ, ∆.

Семантическая таблица <Γ ; ∆> называется выполнимой, если существует такая интерпретация I и такой набор значений d1, d2, . . . , dn ∈ DI свободных переменных, для которых

1. I |= ϕ(x1, x2, . . . , xn)[d1, d2, . . . , dn] для любой формулы ϕ, ϕ ∈ Γ
2. I |≠ ψ(x1, x2, . . . , xn)[d1, d2, . . . , dn] для любой формулы ψ, ψ ∈ ∆.

Теорема (о табличной проверке общезначимости)

|= ϕ ⇐⇒ таблица Tϕ = <∅ ; {ϕ}> невыполнима.

(доказательство по определению)

Закрытая семантическая таблица. Семантическая таблица <Γ ; ∆>, у которой Γ ∩ ∆ ≠ ∅, называется закрытой.

Утверждение

Закрытая таблица невыполнима.

Атомарная семантическая таблица. Семантическая таблица <Γ; ∆>, у которой множества Γ, ∆ состоят только из атомарных формул, называется атомарной.

Утверждение

Незакрытая атомарная таблица выполнима.

Доказательства такого вида (через преобразование таблицы Tϕ = <∅; {ϕ}> к закрытой) называются логическим выводом.

Если в выводе участвуют семантические таблицы, то логический вывод называется табличным семантическим выводом.

Подстановка – это всякое отображение θ: Var → Term, сопоставляющее каждой переменной некоторый терм.

Множество Domθ = {x : θ(x) ≠ x} называется областью подстановки. Если область подстановки – это конечное множество переменных, то такая подстановка называется конечной. Множество конечных подстановок обозначим Subst.

Если θ ∈ Subst и Domθ = {x1, x2, …, xn}, то подстановка θ однозначно определяется множеством пар {x1/θ(x1), x2=θ(x2), …, xn=θ(xn)}.

Каждая пара xi=θ(xi) называется связкой.

Для заданного логического выражения E и подстановки θ запись Eθ обозначает результат применения подстановки к E который определяется так:

Если E = x, x ∈ Var, то Eθ = (x)

Если E = c, c ∈ Const, то Eθ = c

Если E = f(t1, t2, …, tk), то Eθ = f(t1θ, t2θ, …, tnθ)

Если E = P(t1, t2, …, tk), то Eθ = P(t1θ, t2θ, …, tnθ)

Если E = φ&Ψ, то E = φ&Ψ (аналогично для формул φ→Ψ, φvΨ, ¬φ)

Если E = ∀x0 φ, то Eθ = ∀x0(φθ’), где η – новая подстановка, удовлетворяющая условию. Θ’(x) = { x0, если x = x0; θ(x), если x ≠ x0} (аналогично для формул ∃x0 φ).

Переменная x называется свободной для терма t в формуле φ(x), если любое свободное вхождение переменной x в формуле φ(x), не лежит в области действия ни одного квантора, связывающего переменную из множества Var t.

Подстановка θ = {x1/t1, …, xn/tn}, называется правильной для формулы φ, если для любой связки xi/ti переменная xi свободна для терма ti в формуле φ.

**Лекция 4. Подстановки. Табличный вывод. Корректность табличного вывода.**

**Здесь должна быть лекция 4, но из-за лагов с кодировкой у меня ничего не вышло, так что сокращённый вариант – т.е. без формулировок**

Подстановка

Область подстановки

Результат применения подстановки

Переменная, свободная для терма

Подстановка, правильная для формулы

Табличный вывод – это корневое дерево, в котором корнем является сама таблица, дуги выходят в соответствии с правилами табличного вывода, а листьями служат либо закрытые таблицы, либо атомарные таблицы

Успешный табличный вывод – если дерево вывода – конечно и все его листья – закрытые таблицы

Лемма о корректности правил вывода

Каково бы ни было правило табличного вывода, исходная таблица выполнима <=> выполнима таблица, порождённая правилом (одна или обе, если их 2)

(доказательство по определению, для существования доказывается построением новой интерпретации, отличающейся от изначальной (в которой верна первая таблица) лишь равенством константы – подставленному объекту интерпретации)

Теорема корректности табличного вывода

Если для таблицы существует успешный табличный вывод, то она невыполнима

(следует из определения, леммы о корректности правил и утверждения о невыполнимости закрытых таблиц)

Следствие

Если для таблицы T=<∅|ϕ> можно построить успешный табличный вывод, то |= ϕ

**Лекция 5. Полнота табличного вывода. Теорема Левенгейма-Сколема. Теорема компактности Мальцева. Автоматическое доказательство теорем.**

Теорема полноты табличного вывода

Если семантическая таблица T0 невыполнима, то для T0 существует успешный табличный вывод.

(Доказательство конструктивно, в нём указывается, как нужно использовать табличный вывод (Лекция 5 Слайд 3-36) (do not be afraid of the number of pages))

(Доказательство проводится для упрощённой задачи – для конечного числа формул, все они замкнуты и в них нет функциональных символов. Суть в том, что мы последовательно раскрываем вширь дерево табличного вывода нумеруя таблицы, в таблице – упорядоченные списки формул, и с ней ассоциировано множество уже использованных констант. Далее от обратного – пусть нет успешного табличного вывода, тогда либо он бесконечен, либо есть атомарная незакрытая таблица (противоречие очевидно), если бесконечная, то постоим множество объединений всех формул, которые входят в эту цепь и констант, для них есть интерпретация в которой всё это выполнено, потом индуктивно по количеству логических операций доказываем, что любая формула из множества составленного из выполнимой части таблицы будет выполнима в выбранной интерпретации, и из другой части таблицы – невыполнима, а значит таблица выполнима в этой интерпретации – противоречие условию)

(Несмотря на полное указание алгоритма, на основе которого можно строить пруверы, есть всё ещё проблема с тем, что порой очень много проблем с перебором, и для его уменьшения извращаются по-разному, но этот вопрос всегда открыт)

Теорема Геделя (о полноте)

Если формула ϕ общезначима, то существует успешный табличный вывод для таблицы Tϕ = <∅|ϕ>.

(следует из теоремы полноты табличного вывода)

Теорема Левенгейма-Сколема

Формула ϕ выполнима ⇐⇒ ϕ имеет модель с конечной или счетно-бесконечной предметной областью.

(Если ϕ выполнима, то тогда бы нашлась в соответствии с теоремой о полноте табличного вывода либо атомарная незакрытая таблица, либо счётная ветвь в табличном выводе)

Теорема Компактности Мальцева

Γ |= ϕ ⇐⇒ существует такое конечное подмножество Γ’, Γ’⊆ Γ, что Γ’|= ϕ.

(следствие означает соответствующую невыполнимость таблицы, т.е. наличие успешного табличного вывода, т.е. его конечность, т.е. количество формул к которым был применён табличный вывод – конечно, а значит остальное можно обрезать и для нового конечного набора формул – этот табличный вывод останется верным)

**Лекция 6. Общая схема метода резолюций. Равносильные формулы. Теорема о равносильной замене. Предваренная нормальная форма. Сколемовская стандартная форма. Системы дизъюнктов.**

**Здесь должна быть лекция 6, но из-за лагов с кодировкой у меня ничего не вышло, так что сокращённый вариант – т.е. без формулировок**

Общая схема метода резолюций

Резольвента

Пустой дизъюнкт □

Эквивалентность – это лишь оператор в формуле: ϕ≡ψ ⇔ (ϕ→ψ)&(ϕ→ψ)

Равносильные формулы – это формулы для которых верно |= ϕ≡ψ

Теорема о равносильной замене

|= ψ≡χ => |= ϕ[ψ] ≡ ϕ[ψ/χ]

(Доказывается индукцией по числу связок и кванторов в формуле ϕ)

Предварительная нормальная форма (ПНФ) – ϕ= Q1x1Q2x2…Qnxn M(x1, x2, …, xn)

Квантовая приставка

Теорема о ПНФ

Для любой замкнутой формулы существует предварительная нормальная формула

(Переименование переменных, потом удаление импликации, потом продвижение отрицания вглубь, вынесение кванторов наружу, приведение к КНФ)

Сколемовская стандартная форма (ССФ) – ∀1x1∀2x2…∀nxn M(x1, x2, …, xn) (без ∃)

Теорема о ССФ

Для любой замкнутой формулы существует такая сколемовская стандартная формула, что одна выполнима тогда и только тогда, когда выполнима другая

(через лемму об удалении кванторов существования)

Лемма об удалении кванторов существования

Сколемовский терм

Сколемовская константа

Сколемизация

Теорема

Сколемовская стандартная форма невыполнима тогда и только тогда, когда множество формул (все отдельные конъюнкции с кванторами ∀) не имеет модели

Литеры

Невыполнимая, противоречивая система дизъюнктов – система не имеющая ни одной модели, в которой бы выполнялись все дизъюнкты

**Лекция 7. Эрбрановские интерпретации. Теорема Эрбрана. Задача унификации**

Метод резолюций:

Проверка общезначимости формулы ϕ сводится к проверке противоречивости системы дизъюнктов Sϕ.

Этап 1. Сведение проблемы общезначимости к проблеме противоречивости: ϕ -> ϕ0 = ¬ϕ

Этап 2. Построение предваренной нормальной формы (ПНФ). ϕ0 -> ϕ1 = Q1x1Q2x2. . . Qnxn(D1&D2& . . . &DN)

Этап 3. Построение сколемовской стандартной формы (ССФ). ϕ1 -> ϕ2 = ∀xi1∀xi2. . . ∀xik(D1&D2& . . . &DN)

Этап 4. Построение системы дизъюнктов. ϕ2 -> Sϕ = {D1, D2, . . . , DN},

ϕ общезначимая ⇐⇒ система дизъюнктов Sϕ противоречива.

Эрбрановские интерпретации — это специальная разновидность интерпретаций, в основе которых лежат свободные алгебры.

Пусть задана некоторая сигнатура σ = <Const, Func, Pred>. Тогда эрбрановским универсумом σ называется множество термов Hσ, где i = 0 H0 ={Const , если Const ≠ ∅; {c}, если Const = ∅ (эрбрановская константа)}, i → i + 1 Hi +1 = Hi∪ {f(k)(t1, . . . , tk): f(k)∈ Func, t1, . . . , tk ∈ Hi}

Эрбрановский универсум — это множество всех термов, которые можно построить из констант и функциональных символов заданной сигнатуры (т.е. это предметная область эрбрановских интерпретаций). Термы эрбрановского универсума не содержат переменных и называются основными термами.

Эрбрановская H-интерпретация IH = <Hσ, ConstH(с верхн. подчёрк), FuncH(с верхн. подчёрк), Pred(с верхн. подчёрк)> сигнатуры σ = <Const , Func , Pred> состоит из

* стандартной предметной области — эрбрановского универсума Hσ
* стандартной оценки констант: ConstH(c) = c, т. е. значением каждого константного символа c является его собственное изображение
* стандартной оценки функциональных символов: FuncH(f(n)) = f : f (t1, t2, . . . , tn) = f(n)(t1, t2, . . . , tn),т. е. каждый функциональный символ f играет рольконструктора термов эрбрановского универсума
* произвольной оценки предикатных символов.

Теорема о H -интерпретациях

Система дизъюнктов S выполнима тогда и только тогда, когда S имеет эрбрановскую модель, т.е. выполнима хотя бы в одной H -интерпретации.

(В одну сторону очевидно, а в другую – построим гомоморфное отображение)

Следствие

Система дизъюнктов S противоречива тогда и только тогда, когда S невыполнима ни в одной эрбрановской интерпретации.

Пусть P(m) ∈ Pred, и t1, . . . ,tm ∈ Hσ — набор основных термов. Тогда формула P(m)(t1, . . . , tm) называется основным атомом. Множество всех основных атомов называется эрбрановским базисом и обозначается BH. Всякая H-интерпретация I задается подмножеством BI истинных основных атомов: BI= {P(m)(t1, . . . ,tm) : I |= P(m)(t1, . . . ,tm), {t1, . . . , tm} ⊆ H}.

Пусть имеется дизъюнкт D = ∀x1…∀xm (L1 ∨…∨ Lk) и набор основных термов t1, . . . , tm из эрбрановского универсума Hσ. Тогда дизъюнкт D’ = (L1 ∨ · · · ∨ Lk){x1/t1, . . . , xm/tm}, полученный из D подстановкой основных термов t1, …,tm вместо всех переменных дизъюнкта D называется основным примером дизъюнкта D.

Теорема Эрбрана

Система дизъюнктов S = {D1, . . . , DN} противоречива ⇐⇒ существует конечное противоречивое множество G’ основных примеров дизъюнктов системы S.

(система противоречива <=> для каждой H-интерпретации она противоречива, т.е. любой основной пример для некоторого дизьюнкта – ложен, а дальше - опираясь на теорему Компактности Мальцева, т.е. что всегда можно выделить конечное подмножество из множества)

Правило вывода, разрешающее противоречия, называется правилом резолюции. Это правило можно применять до тех пор, пока не возникнет неустранимое противоречие D1=L и D2=¬L. Это и будет служить признаком противоречивости системы S.

Приведение выражений к общему виду называется унификацией.

Подстановка — это отображение θ: Var → Term.

Конечная подстановка θ = {x1/t1, x2/t2, . . . , xn/tn}. Eθ — применение подстановки θ к выражению E.

Пусть θ, η ∈ Subst. Композиция подстановок θη — это подстановка µ, которая определяется следующим соотношением: µ(x) = (xθ)η для любой x ∈ Var.

Утверждение

Пусть θ = {x1/t1, …, xn/tn}, η = {y1/s1, …, ym/sm}. Тогда подстановка µ = {x1/t1η, …, xn/tnη}∪ ({yi/si: yi∉{x1, x2, …, xn}} − {xj/tjη: xj = tjη} ), является композицией θη.

(Возьмём некоторую переменную z и рассмотрим все возможные случаи)

Пусть E1 и E2 — два логических выражения (термы, атомы, формулы и др.) Подстановка θ называется унификатором выражений E1 и E2, если E1θ = E2θ. Подстановка θ называется наиболее общим унификатором(НОУ) выражений E1 и E2, если

1. θ — унификатор выражений E1 и E2
2. для любого унификатора η выражений E1 и E2 существует такая подстановка ρ, для которой верно равенство η = θρ

НОУ(E1, E2) — множество наиболее общих унификаторов выражений E1 и E2.

Задача унификации состоит в том, чтобы для двух выражений E1 и E2 выяснить, являются ли эти выражения унифицируемыми, и, в случае их унифицируемости, вычислить наиболее общий унификатор.

**Лекция 8. Алгоритм Унификации.**

Лемма (о связке)

Пусть x ∈ Var , t ∈ Term. Тогда

1. Если x ∉ Vart, то {x /t} ∈ НОУ(x , t)
2. Если x ∈ Vart и x ≠ t , то НОУ(x , t) = ∅.

(В первом случае проведём рассуждения для любой взятой переменной, а во втором – для любой подстановки длинна терма не сходится)

Подстановка θ называется унификатором системы уравнений E E: {t1 = s1; t2 = s2; …tn = sn} если для любого i , 1 ≤ i ≤ n, термы tiθ и siθ одинаковы. (Фактически, унификатор θ = {x1/r1, …, xk/rk} — это решение системы уравнений E в свободной алгебре термов)

Соответствующим образом определяется и наиболее общий унификатор системы уравнений

Cистема уравнений E называется приведенной, если E: {x1 = s1; x2 = s2; …; xn = sn} и при этом

1. {x1, . . . , xn} ⊆ Var
2. все переменные x1, ..., xn попарно различные
3. {x1, . . . , xn} ∩ = ∅.

Лемма (о приведенной системе)

Если система уравнений E. E: {x1 = s1; x2 = s2; …; xn = sn} является приведенной, то подстановка {x1/s1, x2/s2, …, xn/sn} является наиболее общим унификатором системы E.

(по лемме о связке)

Системы уравнений E1 и E2 называются равносильными, если НОУ(E1) = НОУ(E2).

Описание алгоритма унификации (Алгоритм Мартелли–Монтанари).

Это — недетерминированный алгоритм, состоящий из 6 правил, которые можно применять в любом порядке до тех пор, пока

1. либо ни одно из правил применить невозможно (построена приведенная система уравнений)
2. либо применяется правило, устанавливающее невозможность унификации.

Исходная система E0; i = 0;

while применимо одно из 6 правил do

выбрать правило R , применимое к Ei;

Ei ++ = R (Ei)

od

Правила преобразования решения уравнений.

1. уравнение f (t1’, t2’, …, tk’) = f (s1’, s2’, …, sk’) замещается совокупностью уравнений t1’= s1’, t2’= s2’, …, tk’= sk’
2. если в системе есть уравнение f(t1’, …, tk’) = g(s1’, …, sm’), где f, g ∈ Func ∪ Const , f ≠ g, то система уравнений не имеет решений: СТОП: “Нет унификатора"
3. уравнение s = x, где x ∈ Var, s ∉ Var , замещается уравнением x = s
4. уравнение s = s удаляется из системы
5. если в системе есть уравнение x = s , причем
   * x ∈ Var
   * x ∉ Vars, и
   * переменная x *встречается в каких-либо других уравнениях системы*,

то ко всем *другим* уравнениям системы применяется подстановка {x/s}

1. если в системе есть уравнение x = s , причем x ≠ s , x ∈ Vars, то система уравнений не имеет решений: СТОП: “Нет унификатора".

Теорема (об унификации)

Какова бы ни была система уравнений E.

1. алгоритм унификации Мартелли-Монтанари всегда завершает работу
2. если система уравнений E унифицируема, то в результате работы алгоритма унификации будет построена приведенная система уравнений, равносильная исходной системе E
3. если система уравнений E неунифицируема, то в результате работы алгоритма унификации будет выдано сообщение СТОП: НЕТ УНИФИКАТОРА.

(каждой системе уравнений сопоставляется тройка - <общее число неприведённых переменных, общее число функциональных символов, общее число уравнений>, а теперь разберём все 6 правил алгоритма и докажем, что в 4-х из них, где нет СТОП – характеристика убывает, а делать это бесконечно нельзя, в силу вспомогательной леммы. Так же не забываем доказать корректность – т.е. что после каждого правила система – эквивалентна (упоминая лемму о связке доказываем, что унификатор одной системы является унификатором и другой). + не забываем упомянуть про полноту)

Вспомогательная лемма.

В множестве троек натуральных чисел не существует бесконечно убывающей последовательности <k1, m1, n1> >>> ­<k2, m2, n2> ­>>> … >>> ­ <ki, mi, ni> ­>>> <ki +1, mi +1, ni +1> >>> ­ …

**Лекция 9. Резолютивный вывод. Корректность резолютивного вывода. Применение метода резолюций.**

Пусть задано выражение E и подстановка θ. Подстановка θ: Var → Var называется переименованием, если θ — биекция. Выражение Eθ называется примером выражения E. Если VarEθ = ∅, то пример Eθ называется основным примером выражения E. Если θ — переименование, то пример Eθ называется вариантом выражения E.

Правило резолюции

Пусть D1 = D1’ ∨ L1 и D2 = D2’ ∨ ¬L2 — два дизъюнкта. Пусть θ ∈ НОУ(L1, L2). Тогда дизъюнкт D0 = (D1’∨D2’)θ называется резольвентой дизъюнктов D1 и D2.

Пара литер L1 и ¬L2 называется контрарной парой.

Правило резолюции , θ ∈ НОУ(L1, L2)

Правило склейки.

Пусть D1 = D1’ ∨ L1 ∨ L2 — дизъюнкт. Пусть η ∈ НОУ(L1, L2).

Тогда дизъюнкт D0= (D1’ ∨ L1)η называется склейкой дизъюнкта D1. Пара литер L1 и L2 называется склеиваемой парой.

Правило склейки , η ∈ НОУ(L1, L2)

Определение резолютивного вывода.

Пусть S = {D1, D2, …, DN} — система дизъюнктов. Резолютивным выводом из системы дизъюнктов S называется конечная последовательность дизъюнктов D1’, D2’, …, Di’, Di+1’, …, Dn’, в которой для любого i, 1 ≤ i ≤ n, выполняется одно из трехусловий:

1. либо Di’ — вариант некоторого дизъюнкта из S
2. либо Di’ — резольвента дизъюнктов Dj’ и Dk’, где j , k < i
3. либо Di’ — склейка дизъюнкта Dj’, где j < i.

Дизъюнкты D1’, D2’, …, Dn’ считаются резолютивно выводимыми из системы S.

Резолютивный вывод называется успешным (или, подругому, резолютивным опровержением), если этот вывод оканчивается пустым дизъюнктом □.

Теорема корректности резолютивного вывода

Если из системы дизъюнктов S резолютивно выводим пустой дизъюнкт □, то S — противоречивая система дизъюнктов.

(следует из лемм, доказывающих, что склейка и резольвента являются логическими следствиями)

Лемма 1.

Если D0— резольвента дизъюнктов D1 и D2, то D1, D2|= D0

Лемма 2.

Если D0 — склейка дизъюнкта D, то D |= D0.

**Лекция 10. Полнота резолютивного вывода.**

Теорема о полноте резолютивного вывода.

Если S – противоречивая система дизъюнктов, то из S резолютивно выводим пустой дизъюнкт □.

(По теореме Эрбрана можно свести противоречивую систему дизъюнктов к конечности, потом воспольземся леммой об основных примерах дизъюнктов, а потом перейдём от неё к самим дизъюнктам через лемму о подъёме)

Лемма об основных примерах дизъюнктов.

Если S’ – конечная противоречивая система основных примеров дизъюнктов, то из S’ резолютивно выводим пустой дизъюнкт □.

(По индукции по количеству различных основных атомов в системе. Базис – если нету атомов, а система противоречива, значит она – уже пустой дизъюнкт. Индуктивный переход – возьмём некоторый основной атом, разобьём все дизъюнкты на 3 множества, где этот атом встречается, встречается с отрицанием, и не встречается. Построим все возможные резольвенты и объединим это множество с 3-м упомянутым ранее, - докажем, что оно противоречиво. Возьмём любую интерпретацию, для определённости – наш атом в ней выполнен. Но система дизъюнктов не выполняется, а дальше разбираем варианты, что именно не выполняется в нашей интерпретации. В одном случае – ясно как делать, а во втором случае нужно построить вспомогательную интерпретацию, которая отличается от начальной лишь тем, что выбранный основной атом в ней не выполняется, и рассмотрим теперь все эти множества для новой интерпретации)

Лемма о подъёме.

Пусть D1 и D2 – два дизъюнкта, и при этом Var D1 ∩ Var D2 = ∅

Пусть D1’=D1θ и D2’ =D2θ – два основных примера этих дизъюнктов D1 и D2.

Пусть D0’ – резольвента дизъюнктов D1’ и D2’.

Тогда из дизъюнктов D1 и D2 резолютивно выводим дизъюнкт D0, основным примером которого является D0’.

(Т.к. дизъюнкты не пересекаются, то подстановку можно разделить отдельно для каждого дизъюнкта, литера из контрарной пары имеет свои прообразы (могут быть разные и несколько), которые породили её в качестве части основного примера. Все указанные литеры (отдельно для каждого дизъюнкта) унифицируемы и у них есть НОУ – который является склейкой этих литер. Так же мы запомним тот унификатор, который приводит найденный НОУ к первоначальному, которые при объединении от обоих дизъюнктов дадут унификатор, а значит есть НОУ необходимый для применения резольвентного правила, причём для полученного резольвента основных примеров будет тоже являться примером)

**Лекция 11. Стратегии резолютивного вывода. Вычислительные возможности метода резолюций.**

Стратегия резолютивного вывода – вывод с дополнительными ограничениями на выбор подходящих пар дизъюнктов для получения резольвент.

Стратегия резолютивного вывода называется полной, если она позволяет вывести пустой дизъюнкт □ из любого противоречивого множества дизъюнктов.

Семантическая резолюция – резолюция с правилом I-резолюции – т.е. при построении резольвенты, оба дизъюнкта-предпосылки должны принадлежать разным множествам S1’ и S2’.

Теорема полноты I–резолюции

Если система дизъюнктов S противоречива, то для любой интерпретации I существует успешный I-резолютивный вывод пустого дизъюнкта □ из S .

Входная резолюция

Предположим, что в системе дизъюнктов S выделен некоторый дизъюнкт D0.

Тогда резолютивный вывод пустого дизъюнкта □ из системы дизъюнктов S можно строить, руководствуясь следующими соглашениями:

1. Для построения первой резольвенты D1 выбирается дизъюнкт D0 и некоторый дизъюнкт D ∈ S\{D0};
2. Для построения i-ой резольвенты Di Выбирается резольвента Di−1, построенная на предыдущем шаге вывода, и дизъюнкт D∈S.

Резолютивный вывод такого вида будем называть входным резолютивным выводом, инициированным дизъюнктом D0.

Хорновские дизъюнкты – класс формул, для которых резолютивное опровержение всегда имеет линейную структуру.

Литера L называется положительной, если L — это атом.

Литера L называется отрицательной, если L = ¬A, где A — это атом.

Дизъюнкт D = L1 ∨ L2 ∨ … ∨Ln называется хорновским дизъюнктом (horn clause), если среди литер L1, L2, …, Ln имеется не более одной положительной литеры.

Логические программы. Резолютивное опровержение систем хорновских дизъюнктов — это вычисление ответов на простые запросы, обращенные к базе позитивных знаний. Базы позитивных знаний (хорновские дизъюнкты) становятся, таким образом, логическими программами.

***КОЛЛОКВИУМ КОЛЛОКВИУМ КОЛЛОКВИУМ КОЛЛОКВИУМ КОЛЛОКВИУМ КОЛЛОКВИУМ КОЛЛОКВИУМ***

**Лекция 12. Хорновские логические программы синтаксис. Декларативная семантика логических программ.**

Императивное программирование: программа — это автомат, описывающий последовательности операторов (команд).

Функциональное программирование: программа — это система уравнений, описывающая вычисляемую функцию.

Логическое программирование: программа — это множество формул, описывающих условия решаемой задачи.

Синтаксис логических программ:

«заголовок» ::= «атом»

«тело» ::= «атом» | «тело», «атом»

«правило» ::= «заголовок» ← «тело»;

«факт» ::= «заголовок»;

«утверждение» ::= «правило» | «факт»

«программа» ::= «пусто» | «утверждение» «программа»

«запрос» ::= □ | ? «тело»

Главная особенность логического программирования — полисемантичность: одна и та же логическая программа имеет две равноправные семантики, два смысла.

Программисту важно понимать, ЧТО вычисляет программа. Такое понимание программы называется декларативной семантикой программы.

Компьютеру важно «знать», КАК проводить вычисление программы. Такое понимание программы называется операционной семантикой программы.

Пусть P — логическая программа, D — программное утверждение, а θ — подстановка. Тогда

1. Dθ — пример программного утверждения D,
2. если θ — переименование, то Dθ — вариант программного утверждения D,
3. если VarDθ = ∅, то Dθ — основной пример программного утверждения D,
4. [D] — множество всех основных примеров программного утверждения D,
5. [P] — множество всех основных примеров всех утверждений программы P .

Пусть G =?C1, C2, …, Cm — запрос. Тогда

1. атомы C1, C2, …, Cm называются подцелями запроса G,
2. переменные множества называются целевыми переменными,
3. запрос □ называется пустым запросом,
4. запросы будем также называть целевыми утверждениями.

Логические программы и логические формулы

Каждому утверждению логической программы сопоставим логическую формулу:

1. Правило: D’ = A0 ← A1, A2, …, An

D’ = ∀X1…∀Xk (A1&A2& … &An → A0), где {X1, …, Xk} =

1. Факт: D’’ = A;

D’’ = ∀X1…∀Xk A, где {X1, …, Xk} = VarA

1. Запрос: G = ?C1, C2, …, Cm

G = C1&C2& . . . &Cm

Определение (правильного ответа)

Пусть P — логическая программа, G — запрос к P с множеством целевых переменных Y1, …, Yk.

Тогда всякая подстановка θ = {Y1/t1, …, Yk/tk} называется ответом на запрос G к программе P.

Ответ θ = {Y1/t1, …, Yk/tk} называется правильным ответом на запрос G к программе P, если P |= ∀Z1. . . ∀ZN Gθ, где {Z1, . . . , ZN} = .

Под операционной семантикой понимают правила построения вычислений программы. (Операционная семантика описывает, КАК достигается результат работы программы)

Определение SLD-резолюции (Linear resolution with Selection function for Definite clauses (Р. Ковальски))

Пусть

1. G = ? C1, …, Ci, …, Cm — целевое утверждение, в котором выделена подцель Ci,
2. D’= A0’← A1’, A2’, …, An’ — вариант некоторого программного утверждения, в котором VarG ∩ VarD’ = ∅,
3. θ ∈ НОУ(Ci, A0’) — наиб. общ. унификатор подцели Ci и заголовка программного утверждения A0’.

Тогда запрос G’ = ?(C1, …, Ci−1, A1’, A2’, …, An’, Ci+1, …, Cm)θ

называется SLD-резольвентой программного утверждения D’ и запроса G с выделенной подцелью Ci и унификатором θ.

Определение SLD-резолютивного вычисления

Пусть

1. G0 = ?C1, C2, …, Cm — целевое утверждение,
2. P = {D1, D2, …, DN} — хорновская логическая программа.

Тогда (частичным) SLD-резолютивным вычислением, порожденным запросом G0 к логической программе P называется последовательность троек (конечная или бесконечная) (Dj1, θ1, G1), (Dj2, θ2, G2), …, (Djn, θn, Gn), …, в которой для любого i, i ≥ 1,

1. Dji ∈ P, θi ∈ Subst, Gi— целевое утверждение (запрос);
2. запрос Gi является SLD-резольвентой программного утверждения Dji и запроса Gi−1 с унификатором θi

Частичное SLD-резолютивное вычисление

comp = (Dj1, θ1, G1), (Dj2, θ2, G2), …, (Djk, θn, Gn) называется

1. успешным вычислением (SLD – резолютивным опровержением), если Gn = □;
2. бесконечным вычислением, если comp — это бесконечная последовательность;
3. тупиковым вычислением, если comp — это конечная последовательность, и при этом для выделенной подцели запроса Gn невозможно построить ни одной SLD-резольвенты.

Определение (SLD-резолютивного вычисления)

Пусть

1. G0 = ? C1, C2, …, Cm — целевое утверждение с целевыми переменными Y1, Y2, . . . , Yk,
2. P = {D1, D2, …, DN} — хорновская логическая программа,
3. comp = (Dj1, θ1, G1), (Dj2, θ2, G2), …, (Djn, θn, □) — успешное SLD-резолютивное вычисление, порожденное запросом G к программе P.

Тогда подстановка θ = (θ1 θ2 … θn)|Y1,Y2,...,Yk, представляющая собой композицию всех вычисленных унификаторов θ1, θ2, …, θn, ограниченную целевыми переменными Y1, Y2, …, Yk, называется вычисленным ответом на запрос G0 к программе P.

**Лекция 13. Корректность операционной семантики. Полнота операционной семантики.**

Определение правильного ответа

Пусть P — логическая программа, G — запрос к P с множеством целевых переменных Y1, …, Yk. Тогда всякая подстановка θ = {Y1/t1, …., Yk/tk} называется ответом на запрос G к программе P. Ответ θ = {Y1/t1, …, Yk/tk} называется правильным ответом на запрос G к программе P, если P |= ∀Z1 …∀Z NGθ, где {Z1, …, ZN} =

Определение вычисленного ответа

Пусть

1. G0 = ? C1, C2, …, Cm — целевое утверждение с целевыми переменными Y1, Y2, …, Yk,
2. P = {D1, D2, …, DN} — хорновская логическая программа,
3. comp = (Dj1, θ1, G1), (Dj2, θ2, G2), …, (Djn, θn, □) — успешное SLD-резолютивное вычисление, порожденное запросом G к программе P.

Тогда подстановка θ = (θ1 θ2 … θn)| Y1,Y2,...,Yk, представляющая собой композицию всех вычисленных унификаторов θ1, θ2, …, θn, ограниченную целевыми переменными Y1, Y2, …, Yk, называется вычисленным ответом на запрос G0 к программе P.

Теорема (корректности операционной семантики)

Пусть

1. G0= ? C1, C2, …, Cm — целевое утверждение,
2. P = {D1, D2, …, DN} — хорновская логическая программа,
3. θ — вычисленный ответ на запрос G0 к программе P.

Тогда θ — правильный ответ на запрос G0 к программе P.

(Доказывается индукцией по длине вычисления)

Теорема полноты.

Пусть

1. P = {D1, D2, …, DN} — хорновская логическая программа,
2. G0 =?C1, C2, …, Cm — запрос с множеством целевых переменных Y1, Y2, …, Yk,
3. θ — правильный ответ на запрос G0 к хорновской логической программе P.

Тогда существует такой вычисленный ответ η на запрос G0 к программе P, что θ = ηρ для некоторой подстановки ρ.

(Теорема полноты гласит, что каждый правильный ответ — это пример (частный случай) некоторого вычисленного ответа).

(Рассмотрим любой частный случай нашего правильного ответа (запомним эту подстановку), для него примерим лемму об основных вычислениях, которая даст нам успешное SLD-резолютивное вычисление из множества основных примеров программных утверждений, а потом воспользуемся леммой о подъёме для хорновских дизъюнктов, которая даст нам вычисленный ответ, для которого правильный ответ и запомненная подстановка являются лишь частным случаем, а уже отсюда выведем нужную нам подстановку, являющуюся некоторым вычисленным ответом, применив трюк с подменой всех констант вставленных в используемой подстановке на символы переменных вместо которых они подставлялись)

Лемма об основных вычислениях.

Пусть

1. P = {D1, D2, …, DN} — хорновская логическая программа,
2. G0’ = ?C1’, C2’, …, Cn’ — основной запрос,

и при этом верно P |= G0’.

Тогда существует успешное SLD-резолютивное вычисление запроса G0’, обращенного к множеству [P] основных примеров программных утверждений программы P.

(Множество дизъюнктов из утверждений программы и дизъюнкта, порождённого основным запросом – противоречиво, а значит можно выделить конечное противоречивое множество, а значит по теореме полноты резолюций выводим пустой дизъюнкт. Дальше докажем, что этот вывод является SLD-выводом. Разобьём все дизъюнкты на смешанные и негативные, тогда есть резольвенты 2-х типов - которые попадают в первое множество, или во второе, всегда в выводе найдётся хотя бы одна sld-резольвента (ибо таков пустой дизъюнкт), а потом нужно просто применить трюк, перегруппировав все резольвенты и заменив их на sld ((1+2)+3) == ((1+3)+2) )

Лемма о подъеме (для логических программ)

Пусть G0’ = G0θ’ — основной пример запроса G0 с множеством целевых переменных Y1, …, Ym, обращенный к хорновской логической программе P. Если запрос G0’, обращенный к множеству **основных примеров** программных утверждений [P], имеет успешное вычисление, то исходный запрос G0, обращенный к самой программе P, также имеет успешное вычисление с ответом η, который удовлетворяет равенству θ'=ηρ’ для некоторой подстановки ρ'.

(Доказательство на основе обычной леммы о подъёме, которую применяют много раз подряд)

**Лекция 14. Правила выбора подцелей. Деревья вычислений логических программ. Стратегии вычисления логических программ.**

Определение.

Отображение R, которое сопоставляет каждому непустому запросу G: ?C1, C2, …, Cm одну из подцелей Ci=R(G) в этом запросе, называется правилом выбора подцелей.

Для заданного правила выбора подцелей R вычисление запроса G к логической программе P называется R-вычислением, если на каждом шаге вычисления очередная подцель в запросе выбирается по правилу R.

Ответ, полученный в результате успешного R-вычисления, называется R-вычисленным.

Переключательная лемма

Предположим, что запрос G0 : ?C1, …, Ci, …, Cj, …, Cm к хорновской логической программе P имеет вычисление G0 : ?C1, …, Ci, …, Cj, …, Cm

{ D1: θ1 ∈ НОУ(Ci, D+1) }

G1’ : ?(C1, …, D−1, … Cj, …, Cm)θ1

{ D2: θ2 ∈ НОУ(Cjθ1, D+2) }

G2’ : ?(C1θ1, …, D−1θ1, . . . , D−2, . . . , Cmθ1)θ2

Тогда запрос G0 к программе P также имеет вычисление

(тут вычисление, как и выше, но в другом порядке)

и при этом запросы G2' и G2’’ являются вариантами друг друга, т. е. θ1θ2ρ’ = η1η2 и η1η2ρ''= θ1θ2 для некоторых ρ', ρ'' ∈ Subst.

(лемма говорит, что при изменении порядка выбора подцелей результат вычисления сохраняется (с точностью до переименования переменных))

(Доказательство в лоб – рассмотрением порядка одного и второго)

Теорема сильной полноты

Каково бы ни было правило выбора подцелей R, если θ — правильный ответ на запрос G0 к хорновской логической программе P, то существует такой R-**вычисленный ответ** η, что равенство θ = ηρ выполняется для некоторой подстановки ρ.

(По теореме полноты – существует соответствующий вычисленный ответ, дальше итерационно – на первом шаге мы R-правило дало нам некоторую подцель, мы находим её в нашем вычислении и основываясь на переключательной теореме сдвигаем нужное вычисление вперёд. Остаётся проделать этот трюк лишь нужное число раз)

Деревом SLD-резолютивных вычислений запроса G0 к логической программе P называется помеченное корневое дерево TG0,P, удовлетворяющее следующим требованиям:

1. Корнем дерева является исходный запрос G0
2. Потомками каждой вершины G являются всевозможные SLD-резольвенты запроса G (при фиксированном стандартном правиле выбора подцелей)
3. Листовыми вершинами являются пустые запросы (завершающие успешные вычисления) и запросы, не имеющие SLD-резольвент (завершающие тупиковые вычисления).

Стратегией вычисления запросов к логическим программам называется алгоритм построения (обхода) дерева SLD-резолютивных вычислений TG0,P всякого запроса G0 к произвольной логической программе P.

Стратегия вычислений называется вычислительно полной, если для любого запроса G0 и любой логической программы P эта стратегия строит (обнаруживает) все успешные вычисления запроса G0 к программе P.

(Фактически, стратегия вычисления — это одна стратегий обхода корневого дерева. Как известно, таких стратегий существует много, но среди них выделяются две наиболее характерные: стратегия обхода в ширину и стратегия обхода в глубину с возвратом)

Стратегия обхода в ширину является вычислительно полной.

Стратегия обхода в глубину является вычислительно не полной.

Поскольку соображения эффективности превалируют над требованиями вычислительной полноты, в качестве стандартной стратегии вычисления логических программ была выбрана стратегия обхода в глубину с возвратом. (Программист должен сам позаботиться о надлежащем порядке расположения программных утверждений, чтобы стандартная стратегия вычисления позволяла отыскать все вычисленные ответы.)

**Лекция 15. Алгоритмическая полнота логических программ. Моделирование машин Тьюринга логическими программами. Теорема Черча.**

Тезис Черча

Класс эффективно (алгоритмически) вычислимых арифметических функций в точности совпадает с классом арифметических функций, вычислимых в каждой из перечисленных ниже моделей вычислений

1. машины Тьюринга—Поста,
2. λ-исчисление Черча—Клини,
3. системы равенств Эрбрана—Геделя,
4. алгорифмы Маркова,
5. системы Колмогорова—Шенхаге,
6. машины Минского,
7. ...

Модели вычислений такого вида называются алгоритмически полными.

Машина Тьюринга

Задан ленточный алфавит A = {a0, a1, a2, …, an}, в котором особо выделена одна из букв a0 (пустой символ).

Ленточным словом называется всякое слово в алфавите A. Множество всех ленточных слов обозначим A\*.

Для каждого слова w = z1z2 … zn−1zn будем использовать запись w−1 для обозначения обратного слова znzn−1…z2z1. Задан алфавит состояний Q = {q0, q1, q2, …, qm}, в котором особо выделено одно из состояний q0 (начальное состояние).

Ленточной конфигурацией называется всякое слово вида w'q x w’’, где

1. q — состояние, q ∈ Q,
2. x — ленточная буква, x ∈ A,
3. w', w’’ — ленточные слова, w’, w’’ ∈ A\*.
4. q — это то состояние, в котором находится МТ,
5. x — это буква, которая записана в той ячейке ленты, которую обозревает считывающая головка МТ,
6. w’ — это ленточное слово, составленное из символов, записанных слева от обозреваемой ячейки,
7. w’’ — это ленточное слово, составленное из символов, записанных справа от обозреваемой ячейки.

По умолчанию считается, что во всех остальных ячейках ленты записаны пустые символы.

Ленточная конфигурация вида uq0xv называется начальной конфигурацией.

Множество всех конфигураций обозначим ConfA,Q.

Множество всех начальных конфигураций обозначим Conf0A,Q.

Командой называется всякая пятерка вида q x y q’ D, где

1. q, q’ — состояния, q, q’ ∈ Q,
2. x , y — ленточные буквы, x , y ∈ A,
3. D — направление сдвига головки, D ∈ {L, R}.

Эту команду нужно понимать так: если МТ находится в состоянии q и обозревает символ x, то записать в обозреваемую ячейку символ y, перейти в состояние q’ и сдвинуть считывающую головку на одну ячейку в направлении D.

Каждая команда K задает отношение перехода →K на множестве ленточных конфигураций α →K β

Оно определяется так:

1. если α = uzqxv и K = qxyq’L, то β = uq'zyv ,
2. если α = qxv и K = qxyq’L, то β = q’a0yv ,
3. если α = uqxzv и K = qxyq’R , то β = uyq’zv ,
4. если α = uqx и K = qxyq’R , то β = uyq’a0.

Программа машины Тьюринга — это произвольное множество команд π = {K1, K2, …, KN}. Программа π называется детерминированной, если для любых двух команд этой программы

Ki = qi xyqi’Di , Kj = qj ztqj’Dj, выполняется хотя бы одно из двух условий qi ≠ qj, x ≠ z.

Программа π задает отношение переходов на множестве ленточных конфигураций →π = . Конфигурация α называется заключительной для программы π, если не существует никакой конфигурации β, для которых выполняется α →π β. Это означает, что α = uqxv, и в программе π нет ни одной команды K = qx…

Вычислением машины Тьюринга с программой π на начальной конфигурации α0, α0 ∈ Conf0 называется последовательность конфигураций π(α0) = α0, α1, α2, …, αi, αi+1, … удовлетворяющая следующим условиям:

1. для любого i, i ≥ 0, верно αi →π αi+1;
2. последовательность π(α0) либо является бесконечной, либо заканчивается заключительной конфигурацией αN.

В последнем случае αN называется результатом вычисления. Результат бесконечного вычисления считается неопределенным. Ясно, что вычисление π(α0) детерминированной машины Тьюринга однозначно определяется начальной конфигурацией α0 и программой π.

Лемма 1

Для любой пары ленточных конфигураций α, β ∈ Conf и команды K отношение перехода α →K β выполняется тогда и только тогда, когда запрос Gα: ?P (left (α), right (α), X, Y) и одно из программных утверждений D1 (K), D2 (K) имеют SLD-резольвенту Gβ: ?P (left (β), right (β), X, Y).

(типо очевидно – если есть резольвента – т.е. есть соответствующий НОУ, который описывает лево и право и дальше по построению следует, что выполняется отношение перехода в машине Тьюринга, и наоборот – если выполняется отношение перехода, то мы можем выписать НОУ совершить SLD-резолютивный переход)

Лемма 2

Ленточная конфигурация α ∈ Conf является заключительной для машины Тьюринга π тогда и только тогда, когда запрос Gα: ?P (left (α), right (α), X, Y) и одно из программных утверждений множества имеют пустой дизъюнкт □ в качестве SLD-резольвенты.

(типо так же очевидно как и в предыдущей лемме)

Теорема (о моделировании МТ логическими программами)

Каковы бы ни были машина Тьюринга π и начальная конфигурация α0, вычисление α0 →π α1 →π α2 →π … →π αN завершается заключительной конфигурацией αN тогда и только тогда, когда запрос Gα0: ?R (left (α0), right (α0), X, Y) к хорновской логической программе Pπ имеет успешное вычисление с вычисленным ответом θ = {X/left (αN), Y/right (αN)}.

(следует из предыдущих 2-х лемм)

Теорема Тьюринга об алгоритмической неразрешимости проблемы останова

Проблема останова машин Тьюринга алгоритмически неразрешима, т. е. не существует алгоритма (машины Тьюринга), способного вычислить следующую функцию

(доказательство знает каждый первокурсник)

Следствие 1.

Не существует алгоритма, способного определить по заданному запросу G к хорновской логической программе P,

1. является ли дерево SLD-резолютивных вычислений запроса G конечным;
2. содержит ли дерево SLD-резолютивных вычислений запроса G хотя бы одно успешное вычисление;
3. является ли заданная подстановка θ вычисленным ответом на запрос G .

Следствие 2 (Теорема Черча)

Не существует алгоритма, способного определить по заданной замкнутой формуле логики предикатов ϕ, является ли эта формула общезначимой, т. е. проблема общезначимости "| = ϕ ?"алгоритмически неразрешима.

(просто потому что иначе задача проверки является ли заданная подстановка – вычисленным ответом была бы алгоритмически разрешима, что привело бы к алгоритмической разрешимости задачи останова МТ)

**Лекция 16. Управление вычислениями логических программ. Оператор отсечения.**

Есть два основных способа управления вычислением логической программы:

1. Выбирать правильный порядок расположения атомов в телах процедур (по принципу: вначале решать простые задачи, а потом сложные).
2. Выбирать правильный порядок расположения программных утверждений (по принципу: вначале предлагать простые способы решения, а потом сложные).

Оператор отсечения

Для этого в языках логического программирования вводится оператор отсечения (cut). Этот оператор представляет собой 0-местный предикат **!**, оказывающий специальный побочный эффект.

С точки зрения декларативной семантики, предикат **!** имеет постоянное значение true. Для его описания не требуется никаких программных утверждений (**!** — встроенный предикат). Поэтому оператор отсечения может использоваться в запросах и в телах программных утверждений, не оказывая при этом никакого влияния на их логический смысл.

Операционная семантика оператора отсечения **!** задается следующими правилами:

1. Если запрос G и программное утверждение D: A0 ← A1, …, **!**, …, An порождают SLD-резольвенту G0, то в стеке вычислений программы запрос G получает специальную служебную пометку **(∗**, индивидуальную для каждого вхождения оператора **!**
2. Если в запросе G оператор **!** активен, т. е. G = ? **!**, C1, …, Ck, то в стеке вычислений программы запрос G получает специальную служебную пометку **∗)**, индивидуальную для каждого вхождения оператора **!**, и при этом порождается новый запрос G0 = ? C1, …, Ck;
3. Если по ходу вычисления при откате достигается элемент стека вычислений программы, помеченный **∗)**, то из стека удаляются все элементы, расположенные между элементами, помеченными **(∗** и **∗)** (включая и сами эти элементы).

Программное утверждение A0 ← A1, …, Ak, **!**, Ak+1, …, An содержащее оператор отсечения можно прочитывать двояко:

1. Чтобы решить задачу A0 нужно найти только первое решение задач A1, …, Ak и далее решать задачи Ak+1, …, An. Если решение задач A1, …, Ak найти не удается, то воспользоваться альтернативными процедурами решения задачи A0.
2. Чтобы решить задачу A0 нужно проверить условия A1, …, Ak. Если эти условия выполнены, то приступить к решению задач Ak+1, .., An и не обращаться к другим вариантам решения задачи A0. Если же эти условия не выполнены, то обратиться к альтернативным способам решения задачи A0.

Таким образом, оператор отсечения позволяет удобно использовать в логическом программировании стандартные конструкции императивного программирования.

1. Ветвление. S0: **if** P **then** S1 **else** S2 **fi**

Pif−then−else: S0 ← P, **!**, S1; S0 ← S2;

1. Итерация. S0: **while** P **do** S1 **od**

Pwhile−do: S0 ← P, **!**, S1, S0; S0 ←;

**Лекция 17. Отрицание в логическом программировании. Оператор not. Встроенные предикаты и фукнции. Оператор вычисления значений. Модификация баз данных.**

Допущение Замкнутости Мира. Пусть имеется некоторое непротиворечивое множество замкнутых формул Γ (например, хорновская логическая программа) и замкнутая формула ϕ (например, запрос или отдельная подцель). Тогда формула ¬ϕ является логическим следствием множества Γ в допущении замкнутости мира Γ |=CWA ¬ϕ, если неверно, что ϕ логически следует из Γ, т. е. Γ |≠ ϕ.

(Здесь CWA — аббревиатура Closed World Assumption)

Правило SLDNF-резолюции

Пусть имеется запрос G0: ?not(C0), C1, …, Cn к программе P. Для вычисления SLDNF-резольвенты G1

1. формируется запрос G’: ? C0 к программе P;
2. проводится построение (обход) дерева вычислений T запроса G’: ? C0;
3. в зависимости от устройства дерева T возможен один из трех исходов:
   1. Успех. Дерево T конечно, и все его ветви (SLD-резолютивные вычисления) являются тупиковыми.
   2. Неудача. При построении (обходе) дерева T было обнаружено успешное вычисление.
   3. Бесконечное вычисление. Дерево T бесконечно и при его построении (обходе) не было обнаружено успешных вычислений.

Теорема (корректности SLDNF-резолюции)

Если запрос G: ? not(C0) к хорновской логической программе P имеет успешное SLDNF-резолютивное вычисление, то P |=CWA ¬∃y C0.

(Есть успешное вычисление т.е. древе разбора – все тупики, т.е. не существует такой подстановки, что бы в какой-нибудь интерпретации выводилось бы С0 т.е. из P не следует С0 т.е. по определению CWA следует то, что нужно)

**Лекция 18. Итуиционистская логика.**

Интуиционизм — это философское течение в математике, возникшее в начале 20 века как критический отклик на неограниченное применение формальных логических методов в математике, приводящее к парадоксам (антиномиям). По мнению интуиционистов (Брауэр, Вейль, Пуанкаре), парадоксы возникают в связи с тем, что законы логики, справедливые для конечных множеств, безосновательно переносятся на бесконечные множества.

Семантика Колмогорова–Брауэра–Гейтинга

1. ϕ & ψ: Решить обе задачи ϕ и ψ и предъявить решение;
2. ϕ ∨ ψ: Выбрать одну из двух задач ϕ и ψ, решить выбранную задачу и предъявить решение;
3. ϕ → ψ: Показать, что решение задачи ψ сводится к решению задачи ϕ, т. е. предъявить способ, который позволяет, располагая решением задачи ϕ, построить решение задачи ψ;
4. ¬ ϕ: Доказать, что задача ϕ не имеет решения. Законами интуиционистской логики считаются только те формулы, которые соответствуют описаниям составных задач, имеющих решение при любых условиях.

Законы интуиционистской логики

1. P → P — каждую задачу можно свести к ней самой;
2. (P → Q) & (Q → R) → (P → R) — чтобы свести задачу R к задаче P достаточно найти задачу Q, к которой можно свести задачу R, и которую, в свою очередь, можно свести к задаче P;
3. P → ¬¬P — чтобы убедиться в том, что не существует доказательства неразрешимости задачи P, достаточно найти решение задачи P;
4. (¬P ∨ ¬Q) → ¬(P &Q) — чтобы показать, что обе задачи P и Q нельзя решить одновременно, достаточно выбрать одну из этих задач и показать, что она неразрешима.

Определение (модель Крипке)

Пусть P = {P1, P2, …, Pn, …} — множество атомарных формул (названия задач).

Интуиционистская интерпретация — это реляционная система I = <S, R, ξ>, в которой

1. S ≠ ∅ — множество состояний (состояний знания);
2. R ⊆ S × S — отношение переходов на S, которое является отношением нестрогого частичного порядка:
   1. рефлексивное R(s, s);
   2. транзитивное R(s1, s2)&R (s2, s3) ⇒ R(s1, s3);
   3. антисимметричное R(s1, s2) & R(s2, s1) ⇒ s1 = s2;
3. ξ: S × P → {true, false} — оценка атомарных формул, удовлетворяющая условию монотонности:

R (s1, s2) & ξ (P , s1) = true ⇒ ξ (P , s2) = true.

Определение (семантика Крипке)

Пусть I = <S, R, ξ> — интуиционистская интерпретация. Тогда отношение выполнимости I, s |=I ϕ формулы ϕ в состоянии s интерпретации I определяется так:

1. если ϕ = P ∈ **P**, то I, s |=I ϕ ⇐⇒ ξ (s, P) = true;
2. I, s |=I ϕ1&ϕ2 ⇐⇒ I, s |=I ϕ1 и I, s |=I ϕ2;
3. I, s |=I ϕ1 ∨ ϕ2 ⇐⇒ I, s |=I ϕ1 или I, s |=I ϕ2;
4. I, s |=I ϕ1 → ϕ2 ⇐⇒ для любого состояния s’, если (s, s’) ∈ R и I, s’|=I ϕ1, то I, s’|=I ϕ2;
5. I, s |=I ¬ϕ1 ⇐⇒ для любого состояния s', если(s, s') ∈ R, то I, s' |≠I ϕ1.

Формула ϕ называется интуиционистски общезначимой (законом интуиционистской логики), если для любой интерпретации I и для любого состояния s верно I, s |=I ϕ.

Теорема 1

|=I ϕ ⇒ |=C ϕ

Теорема 2 (дизъюнктивное свойство)

|=I ϕ ∨ ψ ⇐⇒ |=I ϕ или |=I ψ

Теорема 3 (экзистенциальное свойство)

|=I ∀x1 …∀x1∃y ϕ(x1, …, xn, y) ⇐⇒ существует такой терм t(x1, …, xn), что |=I ϕ(x1, …, xn, t(x1, …, xn))

t(x1, ..., xn) — это программа решения задачи ϕ.

Это называется «изоморфизмом Карри–Ховарда».

**Лекция 19. Модальные логики.**

Синтаксис модальных формул

Расширим синтаксис классической логики предикатов, введя два логических оператора:

1. □ (модальность необходимого) и
2. ◊ (модальность возможного),

при помощи которых разрешается строить формулы следующего вида:

1. (□ϕ) «необходимо ϕ»,
2. (◊ϕ) «возможно ϕ».

Семантика Крипке модальных формул

Определим самое общее отношение выполнимости для модальных формул.

Пусть P = {P1, P2, …, Pn, …} — множество атомарных формул (элементарные высказывания).

Модальная интерпретация или модель Крипке — это реляционная система I = <W, R, ξ>, в которой

1. W ≠ ∅ — множество состояний (возможные миры);
2. R ⊆ W × W — отношение достижимости на W,
3. ξ: W × P → {true, false} — оценка атомарных формул.

Система <W, R> называется шкалой Крипке (frame).

Если (w, w') ∈ R, то возможный мир w' называется альтернативным миром для w.

Отношение выполнимости для модальных формул

Пусть I = <W , R, ξ> — модель Крипке.

Тогда отношение выполнимости I, s |= ϕ формулы ϕ в мире s модели I определяется так:

1. если ϕ = P ∈ P , то I, s |= ϕ ⇐⇒ ξ (w, P) = true;
2. I, w |= ϕ1&ϕ2 ⇐⇒ I, w |= ϕ1 и I, w |= ϕ2;
3. I, w |= ϕ1 ∨ ϕ2 ⇐⇒ I, w |= ϕ1 или I, w |= ϕ2;
4. I, w |= ϕ1 → ϕ2 ⇐⇒ I, w |≠ ϕ1 или I, w |= ϕ2;
5. I, w |= ¬ϕ1 ⇐⇒ I, w |= ϕ1;
6. I, w |= □ϕ ⇐⇒ для любого альтернативного мира w' если <w, w’> ∈ R, то I, w'|= ϕ;
7. I, w |= ◊ϕ ⇐⇒ существует такой альтернативный мир w', что <w, w'> ∈ R и I, w' |= ϕ.

Свойства:

1. |= ◊ϕ ≡ ¬□¬ϕ;
2. |= □ (ϕ1 → ϕ2) → (ϕ1 → ϕ2);
3. |= ϕ ⇒ |= □ϕ (правило необходимости).

Характеристические формулы

1. □ϕ → ϕ рефлексивные шкалы ∀w R(w, w);
2. □ϕ → □□ϕ транзитивные шкалы ∀w1∀w2∀w3 (R(w1, w2)&R(w2, w3) → R(w1, w3));
3. ◊□ϕ → □ϕ симметричные шкалы ∀w1∀w2 (R(w1, w2) → R(w2, w1)).

Эпистемические логики и мультагентные системы

Эпистемические логики — это разновидности модальных логик, изучающие модальности знания и мнения (веры) идеализированных агентов. Интерес представляют вопросы о том, какими знаниям располагает субъект, насколько он осознает свои знания (и незнания), и какие причинно-следственные связи возникают между утверждениями, касающимися вопросов знания и веры. В эпистемической логике модальный оператор □ϕ следует прочитывать «Я знаю, что ϕ», а ◊ϕ — «Я допускаю, что ϕ».

Эпистемические логики и мультагентные системы

Основные законы (аксимы) эпистемической логики:

1. Аксиома адекватности знания: □ϕ → ϕ

«Мои знания верны».

1. Аксиома позитивной интроспекции: □ϕ → □□ϕ

«Я вполне представляю все, что мне известно».

1. Аксиома негативной интроспекции: ◊□ϕ → □ϕ

«Я вполне сознаю, что именно мне неизвестно».

Пусть A = {a1, a2, …, an} — множество агентов. Тогда

□a ϕ означает «Агент a знает, что ϕ верно».

□C ϕ означает «Все агенты знают, что ϕ верно».

Темпоральные логики

Темпоральные (временные) логики применяются для описания и исследования причинно-следственных зависимостей, развивающихся во времени. Модальный оператор □ означает «всегда», а оператор ◊ — «когда-нибудь».

2 разновидности темпоральных логик:

1. Логика линейного времени LTL
2. Логика деревьев вычислений CTL (частный случай «Логики ветвящегося времени»), использует темпоральные операторы 2-х типов: универсальные и экзистенциальные ∀□, ∀◊, ∃□, ∃◊.

Логика деревьев вычислений CTL

Пусть I = <S, R, ξ> — древесная модель Крипке для логики CTL, s0 ∈ S — одно из состояний модели. Тогда

1. I, s0|= ∀□ϕ ⇐⇒ в каждом состоянии s, достижимом из состояния s0, верно I, s |= ϕ;
2. I, s0|= ∃□ϕ ⇐⇒ существует ветвь, исходящая из состояния s0, в каждом состоянии s которой верно I, s |= ϕ;
3. I, s0|= ∀◊ϕ ⇐⇒ в каждой ветви, исходящей из состояния s0, есть состояние s, в котором верно I, s |= ϕ;
4. I, s0|= ∃◊ϕ ⇐⇒ существует ветвь, исходящая из состояния s0, в одном из состоянии s которой верно I, s |= ϕ.

**Лекция 20. Правильные программы. Императивные программы. Задача верификации программ. Логика Хоара. Автоматическая проверка правильности программ.**

Описание требований правильности функционирования программы называется спецификацией программы.

Проверка соблюдения вычислениями программы требований правильности функционирования называется верификацией программы.

Определение

присваивание ::= «переменная» ⇐ «терм»

условие ::= «атом» | (¬условие ) | (условие & условие ) | (условие ∨ условие )

программа ::= присваивание | программа ; программа | if «условие» then программа else программа fi | while условие do программа od

Определение (состояния вычисления)

Пусть **Var** — это множество переменных, а GTerm — это множество основных термов сигнатуры σ.

Оценкой переменных (состоянием данных) будем называть всякое отображение (подстановку) θ: **Var** → GTerm.

Состоянием управления будем называть всякую программу, а также специальный символ ∅.

Состоянием вычисления будем называть всякую пару <π, θ>, где π — состояние управления, а θ — оценка переменных.

Запись Stateσ будет обозначать множество всевозможных состояний вычислений сигнатуры σ.

Определение (отношения переходов)

Пусть I — это интерпретация сигнатуры σ. Тогда отношение переходов для императивных программ — это бинарное отношение →I на множестве состояний вычисления Stateσ, удовлетворяющее следующим требованиям:

1. **ASS**: <x ⇒ t, θ> →I <∅, {x/t}θ>;
2. **COMP**\_∅: <π1; π2, θ> →I <π2, η> тогда и только тогда, когда <π1, θ> →I <∅, η>;
3. **COMP**: <π1; π2, θ> →I <π'1; π2, η> тогда и только тогда, когда <π1, θ> →I <π’1, η> и π’1 ≠ ∅;

Определение (отношения переходов)

1. **IF\_1**: <if C then π1 else π2 fi, θ> →I <π1, θ> тогда и только тогда, когда I |= Cθ;
2. **IF\_0**: <if C then π1 else π2 fi, θ> →I <π2, θ> тогда и только тогда, когда I |≠ Cθ;
3. **WHILE\_1**: <while C do π od, θ> →I <π; while C do π od, θ> тогда и только тогда, когда I |= Cθ;
4. **WHILE\_0**: <while C do π od, θ> →I <∅, θ> тогда и только тогда, когда I |≠ Cθ.

Частичным вычислением программы π0 на оценке переменных θ0 в интерпретации I называется последовательность (конечная или бесконечная) состояний вычисления <π0, θ0>, <π1, θ1>, …, <πn−1, θn−1>, <πn, θn>, …, в которой для любого n, n ≥ 1, выполняется отношение <πn−1, θn−1> →I <πn, θn>.

Вычислением программы π0 на оценке переменных θ0 в интерпретации I называется всякое частичное вычисление, которое нельзя продолжить.

Как следует из определения, любое вычисление либо является бесконечной последовательностью, либо завершается состоянием <∅, η>. В последнем случае оценка η называется результатом вычисления.

Будем использовать запись →∗I для обозначения рефлексивного и транзитивного замыкания отношения переходов →I.

Неформальная постановка задачи верификации программ.

Программа π считается (частично) корректной, если для любых начальных данных, удовлетворяющих определенному условию ϕ, результат вычисления (если вычисление завершается) удовлетворяет определенному условию ψ.

Ограничение ϕ, которое налагается на начальные данные, называется предусловием, а требование ψ, которому должны удовлетворять результаты вычисления, называется постусловием программы.

Задача верификации программы π заключается в проверке частичной корректности программы π относительно заданного предусловия ϕ и заданного постусловия ψ.

Определение. Триплетом Хоара (тройкой Хоара) называется всякое выражение вида ϕ{π}ψ, где ϕ, ψ — формулы логики предикатов, а π — императивная программа.

Обозначим HTσ множество триплетов Хоара сигнатуры σ.

Выполнимость триплетов Хоара в интерпретациях определяется так:

I |= ϕ{π}ψ ⇐⇒ для любых оценок переменных θ, η, если I | = ϕθ и <π, θ> →∗I <∅, η>, то I |= ψη.

Определение (частичной корректности программы)

Пусть ϕ, ψ — формулы логики предикатов, а π — императивная программа.

Программа π называется частично корректной в интерпретации I относительно предусловия ϕ и постусловия ψ, если триплет ϕ{π}ψ выполним в интерпретации I, т. е. I |= ϕ{π}ψ.

Правила вывода Хоара

1. ASS: ϕ{x/t} {x ⇐ t} ϕ 🡺 true,
2. CONS: ϕ{π}ψ 🡺 ϕ → ϕ0, ϕ0{π}ψ0, ψ0→ ψ,
3. COMP: ϕ{π1; π2}ψ 🡺 ϕ{π1}χ, χ{π2}ψ,
4. IF: ϕ{if C then π1 else π2 fi}ψ 🡺 (ϕ&C){π1}ψ, (ϕ&¬C){π2}ψ,
5. WHILE: ϕ{while C do π od} (ϕ&¬C) 🡺 (ϕ&C){π}ϕ.

Определение вывода в логике Хоара

Вывод в логике Хоара триплета Φ0 = ϕ0{π0}ψ0 — это корневое дерево, вершинами которого служат триплеты и формулы логики предикатов и при этом

1. корнем дерева является триплет Φ0;
2. из вершины Φi исходят дуги в вершину Φj ⇐⇒ Φi 🡺 Φj — правило табличного вывода;
3. из вершины Φ1 исходят дуги в вершины ϕ1 , Φ3, ϕ2 ⇐⇒ Φ1🡺ϕ1, Φ3, ϕ2 — правило табличного вывода;
4. листьями дерева являются формулы логики предикатов.

Вывод триплета Φ0 = ϕ0{π0}ψ0 в логике Хоара называется успешным в интерпретации I, если дерево вывода является конечным, и все его листовые вершины — это истинные в интерпретации I формулы логики предикатов.

Теорема корректности

Для любой интерпретации I и для любого правила вывода логики Хоара Φ🡺Ψ, Φ🡺ϕ, Φ🡺Ψ1,Ψ2, Φ🡺ϕ,Ψ,ψ, если I |= Ψ, I |= ϕ, , то I |=Ф

(Докажем поочереди все павила ass, cons, comp, if, while. Для ass - возьмём некоторую произвольную оценку переменных. Согласно операционной семантике императивных программ существует единственное вычисление, …)

Следствие.

Если триплет ϕ{π}ψ имеет успешный в интерпретации I вывод, то программа π частично корректна в интерпретации I относительно предусловия ϕ и постусловия ψ.

Полнота правил вывода Хоара

1. Верно ли, что для каждой интерпретации I существует система правил вывода, позволяющая для каждого триплета Φ = ϕ{π}ψ построить успешный вывод Φ в интерпретации I и доказать его успешность в случае I |= Φ?

(Нет, следует из теоремы Гёделя о неполноте)

1. Верно ли, что для каждой интерпретации I существует система правил вывода, позволяющая для каждого триплета Φ = ϕ{π}ψ построить успешный вывод Φ в интерпретации I (но не гарантирующая доказательства его успешности) в случае I |= Φ?

(Может не найтись нужных формул для применения правила Хоара cons, если базовые предикаты сигнатуры будут недостаточно выразительными)

1. Верно ли, что для некоторых интерпретаций I существует система правил вывода Хоара, которая позволяет для каждого триплета Φ = ϕ{π}ψ построить успешный вывод Φ в интерпретации I в случае I |= Φ?

(Да, достаточно чтобы для цикла существовал терм, который для любой оценки переменных был равен n+1 лишь тогда, когда цикл в вычислении совершает n итераций)

Определение

Пусть заданы интерпретация I , императивная программа π и постусловие ψ. Тогда формула ϕ0 называется слабейшим предусловием (weakest postcondition) для программы π и постусловия ψ, если

1. I |= ϕ0{π}ψ,
2. для любой формулы ϕ, если I |= ϕ{π}ψ, то I |= ϕ → ϕ0.

Слабейшее предусловие для программы π и постусловия ψ условимся обозначать wpr (π, ψ).

Теорема

I |= ϕ{π}ψ ⇐⇒

Таким образом, задача построения успешного вывода сводится к задаче вычисления wpr (π, ψ).

Теорема

wpr (x ⇐ t , ψ) = ψ{x/t},

wpr (π1; π2, ψ) = wpr (π1, wpr (π2, ψ)),

wpr (if C then π1 else π2 fi, ψ) =C & wpr (π1, ψ) ∨ ¬C & wpr (π2, ψ)

**Лекция 21. Верификация распределенных программ. Логика линейного времени PLTL. Размеченные системы переходов. Задача верификации моделей программ.**

Такой подход к проверке правильности программ называется верификацией моделей программ (англ. model-checking).

Верификацию распределенных систем нужно автоматизировать. Это можно сделать, например, так.

1. Выбрать логический язык L, на котором можно описывать требования, предъявляемые к программе. Представить эти требования в виде формул ϕ1, …, ϕn.
2. Выбрать математическую модель M, адекватно представляющую все вычисления программы. Модель должна быть устроен так, чтобы каждое вычисление I в модели M являлось интерпретацией языка L.
3. Проверить выполнимость формул ϕ1, …, ϕn на всех вычислениях модели M. Для проверки выполнимости формул языка L на модели программы M должен быть разработан эффективный алгоритм.

Синтаксис PLTL

В PLTL наряду с булевыми логическими связками для описания причинно-следственной зависимости событий во времени применяются темпоральные операторы

1. X (neXttime) «в следующий момент времени»;
2. F (sometime in Future) «когда-то в будущем»;
3. G (Globally) «всегда в будущем»;
4. U (Until) «до тех пор пока»;
5. R (Release) «высвободить, открепить».

Пусть задано множество булевых переменных AP = {p1, p2, …, pn, …} (будем называть их атомарными высказываниями). Формула PLTL — это

* pi, если pi ∈ AP
* (ϕ&ψ), (ϕ∨ψ), (ϕ→ψ), (¬ϕ) если ϕ и ψ — формулы
* (Xϕ), «в следующий момент будет верно ϕ»
* (Fϕ), «когда-то в будущем будет верно ϕ»
* (Gϕ), «всегда верно ϕ»
* (ϕUψ), «ϕ остается верной, пока не станет верной ψ»
* (ϕRψ), «ψ может перестать быть верной только после того, как станет верной ϕ».

Семантика PLTL

Интерпретация PLTL — это темпоральная модель Крипке I = <N, ≤, ξ>, где

1. N = {0, 1, 2, …} — множество моментов времени
2. ≤ — отношение нестрогого линейного порядка на N
3. ξ: N × AP → {true, false} — оценка атомарных высказываний на шкале времени.

Пусть I = <N, ≤, ξ> — темпоральная интерпретация (вычислительная трасса), n ∈ N — момент времени (состояние вычисления), ϕ — формула PLTL.

Тогда отношение выполнимости I, n |= ϕ формулы ϕ в момент времени n в интерпретации I определяется так.

1. Если ϕ = p, p ∈ AP (т. е. ϕ — атомарное высказывание), то I, n |= ϕ ⇐⇒ ξn(p) = true.
2. Если ϕ = ϕ1&ϕ2, то I, n |= ϕ ⇐⇒ I, n |= ϕ1 и I, n |= ϕ2.
3. Аналогично ϕ1 ∨ ϕ2, ϕ1 → ϕ2, ¬ϕ1
4. Если ϕ = Xψ, то I, n |= ϕ ⇐⇒ I, n+1 |= ψ.
5. Если ϕ = Fψ, то I, n |= ϕ ⇐⇒ существует такое k, k≥0, что I, n+k |= ψ.
6. Если ϕ = Gψ, то I, n |= ϕ ⇐⇒ для любого k, k ≥ 0, верно I, n+k |= ψ.
7. Если ϕ = χUψ, то I, n |= ϕ ⇐⇒ существует такое k, k ≥ 0, что I, n+k |= ψ, и для любого i, 0 ≤ i < k , верно I, n+i |= χ.
8. Если ϕ = χRψ, то I, n |= ϕ ⇐⇒ либо для любого k, k ≥ 0, верно I, n+k |= ψ, либо существует такое k, k ≥ 0, что I, n+k |= χ, и для любого i, 0 ≤ i ≤ k, верно I, n+i |= ψ.

Будем называть формулу PLTL ϕ

1. выполнимой в интерпретации I, если верно I, 0|= ϕ (обозначается I |= ϕ);
2. PLTL-общезначимой, если для любой интерпретации I верно I |= ϕ (обозначается |= ϕ).

Законы двойственности.

1. |= ¬Xϕ ≡ X¬ϕ;
2. |= ¬Fϕ ≡ G¬ϕ;
3. |= ¬Gϕ ≡ F¬ϕ;
4. |= ¬(ϕUψ) ≡ ¬ϕR¬ψ;
5. |= ¬(ϕRψ) ≡ ¬ϕU¬ψ.

Законы взаимной зависимости.

1. |= Fϕ ≡ ¬G¬ϕ;
2. |= Gϕ ≡ ¬F¬ϕ;
3. |= ϕUψ ≡ ¬(¬ϕR¬ψ);
4. |= ϕRψ ≡ ¬(¬ϕU¬ψ);
5. |= Fϕ ≡ true Uϕ;
6. |= Gϕ ≡ false Rϕ.

Законы неподвижной точки.

1. |= Fϕ ≡ ϕ ∨ XFϕ;
2. |= Gϕ ≡ ϕ & XGϕ;
3. |= ϕUψ ≡ ψ ∨ (ϕ & X(ϕUψ);
4. |= ϕRψ ≡ ψ & (ϕ ∨ X(ϕRψ).

Определение LTS

Размеченная система переходов (LTS, Labelled Transition System) — это пятерка <AP, S, S0, →, ρ>, в которой

1. AP — множество атомарных высказываний
2. S — непустое множество состояний вычислений
3. S0, S0 ⊆ S, — непустое подмножество начальных состояний
4. →⊆ S × S, — тотальное отношение переходов, тотальность отношения → означает, что для любого состояния s, s ∈ S, существует такое состояние s’, что s → s’ (т. е. из любого состояния можно сделать хотя бы один переход)
5. ρ: S → 2AP — функция разметки, приписывающая каждому состоянию вычислений s, s ∈ S, множество ρ(s), ρ(s) ⊆ AP, всех тех атомарных высказываний, которые являются истинными в состоянии s.

LTS и PLTL-интерпретации

Трассой в LTS M = <AP, S, S0, →, ρ> называется всякая бесконечная последовательность состояний

tr = si0, si1, …, sin, sin+1, …, (∗)

в которой для любого n, n > 0, верно (sin → sin+1). Если si0 — начальное состояние, si0 ∈ S0, то трасса tr называется начальной трассой.

Запись Tr(M) обозначает множество всех трасс LTS M, а запись Tr0(M) — множество всех начальных трасс LTS M.

Каждой трассе tr ∈ Tr(M) вида (∗) сопоставим PLTL-интерпретацию I(tr) = <N, ≤, ξ>, в которой для любого n, n ≥ 0, и p, p ∈ AP, верно соотношение ξ (n, p) = true ⇐⇒ p ∈ ρ(sin).

LTS и распределенные программы

LTS для распределенной системы, состоящей из двух процессов π1 и π2, взаимодействующих посредством разделяемых переменных, строится на основе семантики чередующихся вычислений. Состояниями LTS для системы π1 || π2 объявляются наборы (count1, count2, ξ1, ξ2, χ), где

1. count1, count2 — значения счетчиков команд процессов π1 и π2,
2. ξ1, ξ2 — подстановки, определяющие значения локальных переменных процессов π1 и π2,
3. χ — подстановка, определяющая значения разделяемых переменных.

Задача верификации моделей программ (model checking) для PLTL формулируется так:

для заданной формулы PLTL ϕ и LTS M проверить M |= ϕ.

**Лекция 22. Задача верификации моделей программ. Подформулы Фишера-Ладнера. Табличный метод верификации моделей программ. Алгоритм верификации моделей.**

**Здесь должна быть лекция 22, но из-за лагов с кодировкой у меня ничего не вышло, так что сокращённый вариант – т.е. без формулировок**

Задача верификации моделей программ

Утверждение 1

Для любой LTS M и формула PLTL ϕ верно, что M |≠ ϕ ⇔ существует такая начальная трасса tr, tr ∈ Tr0(M), для которой tr |≠ ϕ.

(следует из задачи верификации моделей программ)

Утверждение 2

В результате применения равносильных преобразований этапа 1 (удаление импликации и темпоральных операторов G, F на основе законов взаимной зависимости), и преобразований этапа 2 (продвижение отрицания в глубь на основании законов двойственности) любая формула PLTL ϕ приводится равносильной формуле ϕ’, представленной в позитивной форме, в которой используются только логические связки и, или, не, и темпоряльные операторы X, F, G, а отрицание применяется только к атомарным высказываниям p ∈ AP.

Подформулы Фишера-Ладнера – FLSubϕ1

Их суть том, что мы вместе с самой PLTL-формулой d позитивной форме включаем ещё и большинство её подформул, на которые она разбирается и для каждого атома, являющегося атомарным высказыванием – мы включаем его отрицание, а для U и R – берём их же в следующий момент времени X, а для самого X – включаем его подформулу.

Утверждение 3

Если ϕ1 содержит n логических связок и темпоральных операторов, то |FLSubϕ1| ≤ 3n

Next-подформулы

XSubϕ1= {ψ : ψ = Xχ, ψ ∈ FLSubϕ1}

(Until-Release)-подформулы

URSubϕ1= {ψ : ψ = χ1Uχ2, ψ ∈ FLSubϕ1} ∪ {ψ : ψ = χ1Rχ2, ψ ∈ FLSubϕ1}

Согласованное множество подформул

Это всякое подмножество множества Фишера-Ладнера FLSubϕ1, которое удовлетворяет условиям – true принадлежит, а false – нет, для любого высказывания выполняется лишь одно из 2-х включений – либо атом, либо его отрицание, для дизъюнкции – или то или другое, для конъюнкции – и то и другое, для U и R – по особенному.

(По сути это множества формул, которые не содержат явных противоречий, т.е. противоречий, которые можно обнаружить в текущий момент времени)

Conϕ1 – совокупность всех возможных согласованных множеств подформул Фишера-Ладнера

Утверждение 4.

Пусть I – произвольная темпоральная интерпретация, и ϕ1 – произвольная формула в позитивной форме.

Тогда для любого момента времени n множество формул Bn = {ψ: ψ ∈ FLSubϕ1 и I, n |= ψ} является согласованным.

(Доказывается непосредственно из определения согласованного множества)

Утверждение 5

Утверждение 6

Если ϕ1 содержит n логических связок и темпоральных операторов, то число различных согласованных множеств подформул Фишера-Ладнера не превосходит величины 23n

Система Хинтики

Это раскрашенный ориентированный граф, в котором вершины – это пары состояние и некоторое согласованное множество, а рёбрами в графе являются те пары, которые позволяют подтвердить все обещания основанные на next-подформулах и выполнить их в следующий момент.

Раскраска идёт по until-release-подформулам (по сути цвет означает, что именно в этой вершине произошёл перескок в правиле U или R)

Бесконечный маршрут в графе называется радужным, если в нём бесконечно часто встречаются вершины каждого цвета.

Основная теорема

Для любой формулы PLTL ϕ1 в позитивной форме и LTS M = <AP, S, S0, ->, ρ>

M |≠ ϕ1 ⬄ в графе Гϕ1,M существует хотя бы один радужный маршрут, исходящий из вершины v0 = (s0, B0), в которой s0 ∈ S0 и ϕ1 ∉ B0

(Доказательство через индукцию)

Ориентированный граф Г называется сильно связным, если для любой пары вершин u и v в графе Г существует маршрут в обоих направлениях.

Всякий максимальный сильно связный подграф графа Г называется компонентой сильной связности.

Компоненту сильной связности будем называть радужной, если в ней содержатся вершины всех цветов.

Теорема

Из вершины v в графе Гϕ1,M исходит радужный маршрут тогда и только тогда, когда существует маршрут, ведущий из вершины v хотя бы в одну из вершины хотя бы одной радужной компоненты сильной связности.

Алгоритм верификации моделей программ

1. Построить равносильную позитивную формулу
2. Построить систему Хинтики
3. Выделить множество подформул until-release и раскрасить граф
4. Выделить радужные компоненты сильной связности
5. Выделить множество всех вершин графа, из которых достижимы радужные компоненты сильной связности
6. Выделить множество всех вершин, которые могут быть начальными
7. Пересечь множества из 5-го и 6-го пункта

**Лекция 23.**

∈∉≠∅θ→Ψλφϕξνλρτχψωδε≡¬v∃∀⇔⇒αβµη∩□◊Qπ≤∪∩

<https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2>