Theories

Theory 6/4/2020

Theory: Σ: 符号 Axiom A: formulas

T_E: Σ: {=, a, b, c, f, g, h, p, q, r}

（常量、函数、谓词）

A: 自反、对称、传递, congruence:

{ $\forall x, y (\bigwedge x_i = y_i) \to f(x) = f(y)$
$\forall x, y (\bigwedge x_i = y_i) \to p(x) \leftrightarrow p(y)$ }

undecidable
无谓词: decidable
complex

T_{PA}: Σ_{PA}: {0, 1, +, ×, =}

A: $\forall x. \neg(x+1) = 0$ (zero)

$\forall x, y (x+1 = y+1 \to x = y)$ (successor)

+0: $x + 0 = x$ $x + (y+1) = (x+y)+1$

×0: $x \times 0 = 0$ $x \times (y+1) = x \times y + x$

数学归纳法: $F(0) \wedge (\forall x. F(x) \to F(x+1)) \to \forall x. F(x)$

undecidable
无谓词也 undecidable, not complete

T_N: Σ_N: {0, 1, +, =}

前没有乘个X公理

⇓

decidable
complete

⇒ 扩到 Z:

{ 乘法: $x \to (x_p - x_n)$
减法: 移到右侧 }

T_Z: Σ_Z: {..., −2x, −1, 0, 1, 2, ..., 3x, 2x, ..., +, −, =, >}

⇓

T_Z 可以转化为 T_N 的等价 not convex

T_A: Σ_A: {=, ·[·], ·⟨·◁·⟩}
读 写

A = { congruence: $\forall a, i, j.\ i = j \to a[i] = a[j]$
read over write 1: $\forall a, v, i, j.\ i = j \to a\langle i \triangleleft v \rangle[j] = v$
2: $\forall a, v, i, j.\ i \neq j \to a\langle i \triangleleft v \rangle[j] = a[j]$ }

undecidable
无谓词: decidable
not convex

⇓

T_E 的 sat 判定: congruence closure (仅考虑 ∧)

仅谓词, 不谓词判定

F: $(s_1 = t_1) \wedge \ldots (s_m = t_m) \wedge$
$(s_{m+1} \neq t_{m+1}) \wedge \ldots (s_n \neq t_n)$

F sat ⟺ ∃ congruence 关系 ~, 使所有 $s_i = t_i$, 所有 $s_i \nsim t_i$

对 S: $S_1 R S_2$ 表示 S_1, S_2 等价关系: 自反, 对称, 传递

congruence: $(\bigwedge s_i R t_i) \to f(s) R f(t)$

等价类 / congruence 类

闭包: 包含... 的最小集合

算法: { ① 构建 $\{s_1 = t_1, \ldots, s_m = t_m\}$ 的 congruence closure
② 若 $s_i \sim t_i$, 对 $(m+1, \ldots, n)$ 成立, unsat
否则 sat }

① 构建 DAG 图

e.g. $f(f(a,b), b)$

构建 subterm set

⇒ {{a}, {b}, {f(a,b)}, {f(f(a,b), b)}}

② 根据 $s_i = t_i$, 合并等价类

闭包

T_A 的 sat 判定 (仅有 ∧): 替换成 T_E 判定

读: $a[i]$ 用 $f_a(i)$ 替代

写: 分支 { $F[v] \wedge i = j$
$F[a[j]] \wedge i \neq j$ }

$F[a\langle i \triangleleft v \rangle[j]]$

每个分支均 unsat 才 unsat
否则 sat

SMT: 扩展到任意

Nelson-Oppen: 不同 theory 结合 ∧ 的方法

① purity: 将公式分为 T_1 的 F_1 与 T_2 的 F_2

② equate: F_1 与 F_2 相关变量连接

① purity: ① 仅含 T_1, T_2 符号

② $f(\ldots t \ldots)$, $f \in \Sigma_i$, $t \notin \Sigma_i$
或 $p(\ldots t \ldots)$, $p \in \Sigma_i$, $t \notin \Sigma_i$ 用 w 替 t, 加上 $t = w$

对于 shared variables:

F sat ⟺ ∃α(V, E),

{ $F_1 \wedge \alpha(V, E)$ T_1 sat
$F_2 \wedge \alpha(V, E)$ T_2 sat }

① guess and check
查所有 shared variable 的 partition,
看是否可能使 F_1, F_2 均 sat 的
低效

② propagation:
不断寻找 implied equalities
e.g. $u = v$
{ convex: 前兆 $u = v$ 加入 F_1 或 F_2
non-convex: 将 $u_i = v_i$ 每个分支加入 F_i }

形成 DPLL 分叉

⇓

每个分支都存在 unsat
否则 sat

SMT: 扩展到任意布尔

核心: 仅能判定有 ∧ 的一阶逻辑
命题逻辑

Naive:

① 将每个一阶 formula 转化为 Boolean Abstraction, 作为 $P_1 \ldots P_n$ 命题公式 $B(F)$:

F sat ⇒ B(F) sat: 必要条件 ⇒ 命题逻辑

B(F) unsat ⇒ return unsat

② 找到一个 α for B(F), 构造出对应的公式 C ($P_1 \wedge \neg P_2 \wedge P_3 \wedge \neg P_4$ 之类).

用 T-solver 判定 $B^{-1}(C)$ ⇒ 仅含 ∧ 的一阶逻辑 ⇒ 向左 ∧ 个 T 判定

③ 若为 sat
否则 F 上加入 ¬C (类似 conflict clause)

⇓

unsat cores:

$C^* \in C$,

{ C^* unsat
去掉 C^* 任一子项, C^* sat } ⇒ 在 F 上加入 $\neg C^*$ 而非 $\neg C$

与 CDCL 结合:

{ theory propagation
backjump }

⇓

theory propagation:

在 propagation 时进行分析
得到 theory lemmas

⇓

加入

(exhaustive, heuristic 启发)

谓词: $p(t_1, \ldots t_n) \Leftrightarrow f_p(t_1, \ldots t_n) = 1$

命题逻辑 Propositional logic | Analyze / Validate 图表 6/3/2020 | 谓词逻辑(一阶) First order logic

元素

atom: T, ⊥, P, Q ...(变量)
literal: atom / ¬atom
formula: ¬, ∧, ∨, →, ↔ 连接

logical symbol (命题逻辑运算符) + ∃, ∀
non-logical { constants: c_1, c_2 ... ; function: f, g, h ; predicate: p, q, r (可以理解为[illegible]) }

terms: 变量、常量、$f(t_1, \ldots, t_n)$
⇒ atoms: ⊥, T, $p(t_1, \ldots, t_n)$
literal: atom / ¬atom
formula: 用 literal 用 逻辑 ∀, ∃ 连接
∀x. F[x]: x: 约束变元 (bound / free)
scope
close formula: 全约束
open formula: 有自由变元 (free: 无变量)

语义 semantics

Interpretation I: variable 真假值
F + I ⇒ 真值
{ satisfying: I ⊨ F ; falsifying: I ⊭ F }
satisfiable: ∃I, I ⊨ F
valid: ∀I, I ⊨ F (F valid ⇔ ¬F unsatisfiable)
等价: $F_1 ⇔ F_2$: $F_1 ↔ F_2$ valid | implication: $F_1 ⇒ F_2$: $F_1 → F_2$ valid
equisatisfiability: F 可满足 ⇔ G 可满足 (弱于等价)

D: 论域
I: interpretation { constant ; function ; predication }
α: assignment: 变量 → D 中值
D + I + α + F ⇒ 真值 { D, I, α ⊨ F ; D, I, α ⊭ F }
⇓
有全称:
D, I, α ⊨ ∀x. F ⇔ ∀c ∈ D, D, I, α[x ↦ c] ⊨ F
D, I, α ⊨ ∃x. F ⇔ ∃c ∈ D, D, I, α[x ↦ c] ⊨ F

valid: ∀D, I, α, D, I, α ⊨ F
satisfiable: ∃ D, I, α, D, I, α ⊨ F
(F valid ⇔ ¬F unsat)

prove valid

① search: 遍历真值表
※ ② 推理法:
(1) 假设 invalid: ∃I, I ⊭ F
(2) 推断规则: $\frac{I \models \neg F}{I \not\models F}$, $\frac{I \models F \wedge G}{I \models F, I \models G}$, $\frac{I \models F \vee G}{I \models F \mid I \models G}$, $\frac{I \not\models F \wedge G}{I \not\models F \mid I \not\models G}$, $\frac{I \models F \leftrightarrow G}{I \models F \wedge G \mid I \models \neg F \wedge \neg G}$
(3) every branch: contradiction ⇒ valid
$\frac{I \not\models F \leftrightarrow G}{I \models F \wedge \neg G \mid I \models \neg F \wedge G}$
sound / complete

不能 search, 只能 推理
(1) 假设 invalid
(2) 规则: 命题的 +
① $\frac{D, I, \alpha \models \forall x. F}{D, I, \alpha[x \mapsto c] \models F}$ for any c ∈ D (c: 已有/fresh)
② $\frac{D, I, \alpha \not\models \exists x. F}{D, I, \alpha[x \mapsto c] \not\models F}$ fresh c ∈ D (c: any)
(3) every branch contradiction ⇒ valid
③ 矛盾: $D, I, \alpha[\ldots] \models p(s_1, \ldots, s_n)$, $D, I, \alpha[\ldots] \not\models p(t_1, \ldots, t_n)$
$\forall 1 \le i \le n$, $\langle I, \alpha[\ldots] \rangle(s_i) = \langle I, \alpha[\ldots] \rangle(t_i)$

judge sat

① 范式: NNF: ¬, ∧, ∨, ¬ 在 literals only
DNF: 内 ∧ 外 ∨
※ CNF: 内 ∨ 外 ∧ ⇒ $C_1 \wedge \ldots \wedge C_n$ ($C_i[p]$: p 在 C_i 中出现; $C_i[\neg p]$: p 在 C_i 中取反)
Convert: Tseitin 变换:
① 对每个 non-atomic subformula 用 $T_1 \leftrightarrow P \to T_2$
② 每个子式 ⇒ 变为 CNF F_1, F_2 ...
③ $T_1 \wedge F_1 \wedge F_2$
等价 (可满足)

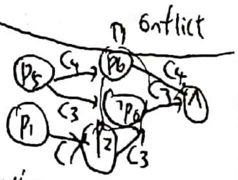
Not decidable (semi-decidable) { valid: 可以停止 ; invalid: 可能不停止 }

② CNF sat 判断
(1) 归结 resolution:
$C_1[p]$ $C_2[\neg p]$ ⇒ 取 p / 取 ¬p
$C_1[p \mapsto \bot] \vee C_2[\neg p \mapsto \bot]$
⇓
resolvent: $C_1[p \mapsto \bot] \vee C_2[\neg p \mapsto \bot]$
e.g.: A∨B∨C 与 ¬A∨B∨D
⇓
B∨C∨D
⇒ 反复归结直至⊥ / 推不出新的结果了

(2) 搜索: search (DPLL)
引入 partial interpretation: (true, false, undef)
C: true: I ⊨ C
false: I ⊭ C
unit: C = C′∨L, I ⊭ C′, L undef

(3) DPLL
decision (手动定的)
[I] ⊭ → [I, p] ⊭ (p 在 F, ¬p 在 I)
backtrack:
$\langle I_1, p, I_2 \rangle ⊭ → [I_1, \neg p] ⊭$ ($[I_1, p, I_2]$ 冲突, p 已被否)
⇒ propagate:
I ⊭ C
C 中除了一个 p 均 false ⇒ 定 p
DPLL: 先 decide, 后 propagate, 最后 backtrack
⇓
全部 回溯 或 结束, 找不到 decide 时 满足

(4) CDCL: 处理冲突, 新增 冲突 子句
左: P 的产生
右: propagate 的 子



找切割 (冲突定义):
decision 在左
至少一个 conflict 在右 ($P_6, \neg P_6$)
⇓
切口 左侧元素 取 ¬, ∨ 上 ⇒ 冲突子式

backjump: 只保留 冲突子式 里面 最早的 decision 在 变换 I 加入, 保留 C

backjump: (红字)
$\langle I_1, p, I_2 \rangle$ ⊭
⇓
$\langle I_1, L \rangle$ ⊭
(C → L) 冲突
C ⊨ I_1
L ∉ I_1
反例: C 的每个子 决策都保留 到最后
加上 L

n: 整数，x: 变元
Aexp: 算术表达式 (+, ×, −)
Bexp: 布尔表达式
C: skip，赋值，$c_1;c_2$，if，while

环境 σ: 变量到值
$\sigma = [x \mapsto 5, y \mapsto 7, z \mapsto 0]$
σ(x)=5 状态
状态: ⟨c, σ⟩ 语句 + 环境
语义: $A[\![a]\!]\sigma$
代入: $A(a[y \mapsto a_0])\sigma = A[\![a]\!]\sigma[y \mapsto A(a_0)\sigma]$

Program Semantics
6/16/2020

小步语义 / 大步语义 / 公理语义
⟨c, σ⟩ → ⟨c′, σ′⟩

Asgn: $\langle x := a, \sigma\rangle \to \langle skip, \sigma[x \mapsto A(a)\sigma]\rangle$
skip: /
seq1: $\dfrac{\langle c_1,\sigma\rangle \to \langle c_1',\sigma'\rangle}{\langle c_1;c_2,\sigma\rangle \to \langle c_1';c_2,\sigma'\rangle}$
seq2: $\langle skip;c,\sigma\rangle \to \langle c,\sigma\rangle$
iftrue: $B(b)\sigma = true$: $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma\rangle \to \langle c_1, \sigma\rangle$
iffalse: $B(b)\sigma = false$: $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma\rangle \to \langle c_2, \sigma\rangle$
while: $\langle \text{while } b \text{ do } c, \sigma\rangle \to \langle \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else } skip, \sigma\rangle$

操作语义 与 ⟶ 一对一
语义函数: $S_{ss}(c)\sigma = \sigma'$ 终止时

命令 c 大步语义 (条件) 写法:
⟨c, σ⟩ ⇓ σ′
Asgn: $\langle x := a, \sigma\rangle \Downarrow \sigma[x \mapsto A(a)\sigma]$
skip: $\langle skip, \sigma\rangle \Downarrow \sigma$
Seq: $\dfrac{\langle c_1,\sigma\rangle \Downarrow \sigma' \quad \langle c_2,\sigma'\rangle \Downarrow \sigma''}{\langle c_1;c_2,\sigma\rangle \Downarrow \sigma''}$
iftrue: $\dfrac{\langle c_1,\sigma\rangle \Downarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2,\sigma\rangle \Downarrow \sigma'}$
iffalse: $\dfrac{\langle c_2,\sigma\rangle \Downarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2,\sigma\rangle \Downarrow \sigma'}$
while false: $\langle \text{while } b \text{ do } c, \sigma\rangle \Downarrow \sigma$
while true: $\dfrac{\langle c,\sigma\rangle \Downarrow \sigma' \quad \langle \text{while } b \text{ do } c,\sigma'\rangle \Downarrow \sigma''}{\langle \text{while } b \text{ do } c,\sigma\rangle \Downarrow \sigma''}$
语义函数 $S_{ns}(c)\sigma = \sigma'$ 终止时

{P} c {Q} Hoare 三元组
前条件 precondition，postcondition
{P} c {Q}: 部分正确，若终止则满足 Q
[P] c [Q]: 全正确，一定终止且满足 Q

skip: $\{P\}\ skip\ \{P\}$
Asgn: $\{Q[a/x]\}\ x := a\ \{Q\}$
Q[a/x]: 用 a 代替 x
例: $[(x+1)/x]: 5 + (x+1)$
Conseq: $\dfrac{P \Rightarrow P' \quad \vdash \{P'\}c\{Q'\} \quad Q' \Rightarrow Q}{\{P\}c\{Q\}}$ 前加强，后减弱
Seq: $\dfrac{\{P\}c_1\{P'\} \quad \{P'\}c_2\{Q\}}{\{P\}c_1;c_2\{Q\}}$
if: $\dfrac{\{P \wedge b\}c_1\{Q\} \quad \{P \wedge \neg b\}c_2\{Q\}}{\{P\}\text{if } b \text{ then } c_1 \text{ else } c_2\{Q\}}$
while: $\dfrac{\{P \wedge b\}c\{P\}}{\{P\}\text{while } b \text{ do } c\{P \wedge \neg b\}}$
P: 循环不变式，进入循环前成立，迭代后仍成立
{P∧b} c {P}
Sound，Relative Complete
推导: 从后向前推

①: 拆分程序循环，切割为 basic path
开始: precondition，loop invariant
结束: loop invariant，assertion，postcondition
引入 assume: 基于到假设
有循环则转为 2 个 basic path
①: precondition: 循环前的 path 及 postcondition
②: 循环体，assume 后 postcondition

② 求 wlp，VC，代入 SMT
若 $pre \to wlp \wedge VC$ valid，则成立
wp: $[wp(c,Q)]\,c\,[Q]$ valid，wp 最弱
wlp: $\{wlp(c,Q)\}\,c\,\{Q\}$ valid，wlp 最弱
sp: $\{P\}\,c\,\{sp(c,P)\}$ valid，sp 最强
规则: Asgn: $wlp(x := a, Q) = Q[a/x]$ (用 a 替 x)
这里 x 需 free box (当 x 的...)
AsgnArr: $\{Q[x\langle a_1 \triangleleft a_2\rangle/x]\}\ x[a_1] := a_2\ \{Q\}$
用 $x\langle a_1 \triangleleft a_2\rangle$ 替 x

Seq: $wlp(c_1;c_2,Q) = wlp(c_1, wlp(c_2,Q))$
VC: $VC(x := a, Q) = \varnothing$
$VC(c_1;c_2, Q) = VC(c_1, wlp(c_2,Q)) \cup VC(c_2,Q)$
$\cup\, VC(c_1, I) \cup VC(c_2, Q)$
if: $wlp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q)$
$= (b \to wlp(c_1,Q)) \wedge (\neg b \to wlp(c_2,Q))$
assume: $wlp(assume\ b, Q) = b \to Q$ (后置条件)
while: wlp 是循环不变式 I
while b do {I} c, Q:
VC: $\{I \wedge \neg b \Rightarrow Q,\ I \wedge b \Rightarrow wlp(c, I)\}$
定理: wlp 和 VC valid: valid
valid → wlp 和 VC 不一定
每个 basic path 的 VC: $P \to wlp(c,Q) \wedge VC$

③: 停止性判断 Total Correctness
保证程序终止
良基关系: well-founded relation
不存在 $s_1 \succ s_2 \succ \dots$ 无穷序列
有 $s_i < s_j$ (即 P 存在最小元)
字典序: $(s_1 \dots s_n) < (t_1 \dots t_n)$: $s_1 \dots s_{i-1}$ 相同，且 $s_i < t_i$
(字典序比较，前 i−1 个一样，且 $s_i < t_i$，有 $s < t$)

复现 VC: 求 wlp 产生的 VC (while)，各 basic path 的 VC，总结出的 VC，代入 SMT 判 valid
注: assert / assume，invariants，pre，post，排名函数 (秩)

对每个循环定义秩函数，将 state 映射到良基关系 S
终止性: 证明每个 basic path，秩递减
total correctness 用
S 满足良基性，秩函数: δ(x)
VC: $p \to wlp(c_1 \dots c_n, \delta(x) < \delta(x)[x'/x])$

1. ①: basic path 对于 while:
(1): {precondition} ... {invariant} ⇒ 不仅保证 invariant
invariant: {P∧b} c {P} 前提 pre，post 条件
(2) {invariant} assume 循环体 {invariant}
(3) {invariant} assume ¬b ... {post}
证明: 求 basic path 的 VC 证 valid

2. 求排名函数: 有待定系数: 前/后
x := 1
while x > 0 do
  if N < 100 then
    N := N+1
    X := X+1
  else N := N+0
    X := X−1
end

设 $\delta(X,N) = aX + bN + c$
有: 若 $X > 0$，$N \le 111$，$aX + bN + c \ge 0$
$aX + bN + c \ge a(X+1) + b(N+1) + c$
$aX + bN + c \ge a(X-1) + b(N+0) + c$
有 $a = 21$，$b = -2$，$c \ge 222$
$\delta(X,N) = 21X - 2N + 222$

4/24/2020

6/8/2020

1. prove validity of $\vdash a\langle i\rangle = v \rightarrow (\forall j.\ a\langle i \triangleleft v\rangle[j] = a[j])$

1	$I, \alpha \not\models F$	
2	$I, \alpha \models a\langle i\rangle = v$	1, →
3	$I, \alpha \not\models \forall j.\ a\langle i \triangleleft v\rangle[j] = a[j]$	1, →
4	$I, \alpha[j \mapsto c_j] \not\models a\langle i \triangleleft v\rangle[j] = a[j]$	3, ∀
5	$I, \alpha[j \mapsto c_j] \models a\langle i \triangleleft v\rangle[j] \neq a[j]$	4, ¬
6	$I, \alpha[j \mapsto c_j] \models i = j$	5. r.o.w.2
7	$I, \alpha[j \mapsto c_j] \models a\langle i \triangleleft v\rangle[j] = v$	6. r.o.w.1
8	$I, \alpha[j \mapsto c_j] \models a[i] = a[j]$	6. congruence
9	$I, \alpha[j \mapsto c_j] \models v = a[i]$	2. sym
10	$I, \alpha[j \mapsto c_j] \models v = a[j]$	8. 9. trans
11	$I, \alpha[j \mapsto c_j] \models a\langle i \triangleleft v\rangle[j] = a[j]$	7, 10, trans
12	$I, \alpha[j \mapsto c_j] \models \bot$	3, 11

优先去∃, ∀

row2: $i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] = a[j]$

逆否命题: $a\langle i \triangleleft v\rangle[j] \neq a[j] \rightarrow i = j$

⇒ 用 row1

2. $a\langle i\rangle = x \wedge x = y \rightarrow a\langle i \triangleleft y\rangle = a$ (×) 有 valid

= 不能作用在整个 array 上

3. 若 ϕ 既是 Σ_Z 也是 $\Sigma_{\mathbb{N}}$, 则 ϕ T_Z-valid ⟺ $T_{\mathbb{N}}$ valid (×)

$\neg(x+1) = 0$ ⇒ T_Z 与 $T_{\mathbb{N}}$ 关系注意 T_Z 有负数, × $T_{\mathbb{N}}$: valid

4. $a\langle i\rangle = e \rightarrow a\langle i \triangleleft e\rangle = a$ 属于 T_E (✓) T_E 可以看作一阶逻辑

5. 大/小步语义证明 ① 对 $\forall c, \sigma, \sigma', s$, 有 $\langle \sigma, \text{break}; c\rangle \Downarrow \langle \sigma', s\rangle$ 则必有 $\sigma = \sigma'$ ⇒ 归纳推导出所有情况 (未给出的不能推出)

规则推导 seqbreak
$$\frac{(\text{break})\ \langle \sigma, \text{break}\rangle \Downarrow \langle \sigma, \text{brk}\rangle}{\langle \sigma, \text{break}; c\rangle \Downarrow \langle \sigma, s\rangle}$$
则有 $\langle \sigma, \text{brk}\rangle = \langle \sigma', s\rangle$, $\sigma = \sigma'$

② 对 $\forall c, \sigma, \sigma', b$, 若 $\langle \sigma, \text{while } b \text{ do } c \text{ end}\rangle \Downarrow \langle \sigma', D\rangle$, $B[\![b]\!]_{\sigma'} = T$, 则存在 $\langle \sigma'', c\rangle \Downarrow \langle \sigma', \text{brk}\rangle$

归纳法: 步数对 while 归纳 (证明时指定)

① 高度为 1: 1 步推出, 分为 while false 及 while true 2

若 while false, $\dfrac{B[\![b]\!]_\sigma = \bot}{\langle \sigma \text{ while } b \text{ do } c \text{ end}\rangle \Downarrow \langle \sigma', D\rangle}$, 有 $\sigma = \sigma'$, 而 $B[\![b]\!]_{\sigma'} = B[\![b]\!]_\sigma = \bot$ 矛盾

只可 while true 2, 则有 $\dfrac{B[\![b]\!]_\sigma = T,\ \langle \sigma, c\rangle \Downarrow \langle \sigma', \text{brk}\rangle}{\langle \sigma \text{ while } b \text{ do } c \text{ end}\rangle \Downarrow \langle \sigma', D\rangle}$, 有 $\sigma'' = \sigma$ 成立

② 若对 n 高度为 n 的推导成立过程, 则对高度为 n+1 的, 其第一步的推导为

while true 1 $\dfrac{B[\![b]\!]_\sigma = T\quad \langle \sigma, c\rangle \Downarrow \langle \sigma', D\rangle}{\langle \sigma', \text{while } b \text{ do } c \text{ end}\rangle \Downarrow \langle \sigma, s\rangle}$,

高为 n 步推出, 即成立

这样 任意 $\langle \sigma, s\rangle$ 可以在经过 n 步推出的情况下, 存在 σ'' 符合条件

DB

4/24/2020

1. select c.name
from (instructor natural join department) as c ✓ 换名

2. group by 注意
select avg(salary), dept_name, ~~name~~ ⇒ 只有被 group by 后 [illegible] 可以被选

group by dept_name

3. 聚集的判断条件
只能 group by 后 + having

分析验证：
6/7/2020

1. 求 wlp(while x>0 do x:=x+1 end, x≤0)
不需要用 invariant, 直接
先猜出为 T, 然后代入验证

T: 若一开始 x≤0: 不进入循环, 程序后必有 x≤0 [illegible] T 为 precondition
一开始 x>0: 不终止
而 ∀P, P→T, T 为 wlp

2. Hoare rules: havoc: Asgn
$$\frac{}{\{\forall a.\,Q(a/x)\}\ havoc\ x\ \{Q\}}$$
(大步式 [illegible] havoc x 可以 $(x \mapsto n)$, n 任意)

repeat:
$$\frac{\{P\}c\{Q\}\quad \{Q\wedge b\}c\{Q\}}{\{P\}\ repeat\ c\ until\ b\ end\ \{Q\wedge \neg b\}}$$
repeat: c; while b do c
将 P Q 两代入 while 条件

hoare 逻辑: 前面的条件, 后面的单一

assert: 必须成立
$$\frac{}{\{P\wedge b\}\ assert\ b\ \{P\}}$$
assert 前置条件 b 必须成立

assume: 假设条件包括在前 (回 x)
$$\frac{b\to P}{\{[illegible]\}\ assume\ b\ \{P\}}$$
不需要 b 必须成立, 只要 b ⊆ P 即可
e.g. if x≥a ⇒ assume x≥a
假设 x 可≥a 求 a

DB query/optimize 6/12/2020

① parsing + translation
② optimization
③ evaluation

selection

File scan:

A1: 线性搜索 block — $b_r \cdot t_T + t_s$ (最坏), $\frac{1}{2} b_r \cdot t_T + t_s$ (平均)

index scan:

A2: primary index, equality on key: 找一个, $(h_i+1) \cdot (t_T + t_s)$

A3: primary index, equality on nonkey: 找多个 (记录 — 不 unique)

$h_i(t_T+t_s) + t_s + b \cdot t_T$

A4: secondary index, equality (candidate key: 一个) $(h_i+1)(t_T+t_s)$

(不是 key: 多个) $(h_i+n) \cdot (t_T+t_s)$

A5: primary key, comparisons

A6: secondary index, comparison: 同 A4 nonkey

Conjunction: $\sigma_{\theta_1 \wedge \theta_2 \dots \theta_n}(r)$

A7: 用单一 index 先选出 θ_i ~ θ_n 中的一个，再逐个检验其他条件

A8: 如果有合适的 composite index

A9: 指针交 (用 record pointer 和多个 index): 指针求交 (index 的指针)

disjunct: $\sigma_{\theta_1 \vee \theta_2 \dots \theta_n}(r)$

A10: 指针求并 (全有 indice 时): 指针并

否则 linear scan

$\neg \sigma_\theta(r)$: 用线性扫描, 若有 index

equivalence rules

1. selection conjunctive
$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$

2. selection 交换
$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$

3. 投影, $\Pi_{L_1}(\dots \Pi_{L_n}(E)) \dots = \Pi_{L_1}(E)$

4. selection join: $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$
$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5. join 可交换 $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$

6. join 可结合 (a) natural: $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
(b) theta: $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$

7. join 可分配 select: (a) $\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$ (θ_0 仅含 E_1)
(b) $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$

8. join 投影分配: (a) $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$
(b) $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$ (θ 中 L_3, L_4)

~~9, 10, 11, 12~~

9, 10: 集合

11. 选择对 ∪ ∩ − 可分配 $\sigma_\theta(E_1 \cup E_2) = \sigma_\theta(E_1) \cup \sigma_\theta(E_2)$

12. 投影仅对 ∪ 分配: $\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$

其他: [illegible]

执行策略

(materialization) [illegible]

① materialize: 自底向上, 每次生成中间结果 — double buffering (2 个缓存 overlap)

② pipelining (不适用 sort, hash-join 等): demand driven: 按需 open() next() close()

producer driven: 不断放入 buffer, 读取 buffer

时间估计: t_T (block transfer) 忽略 CPU, t_s (block seek)

Sort: external sort merge

① Create sort runs: 一次读入 M block 内存中排序, 写回, $\lceil \frac{b_r}{M} \rceil$ 个 runs (b_r)

② merge runs: 用 M−1 block 放 input, 1 个 block output: 每次选最小的写入 output 满后写入 disk

总归并趟数 每一趟 runs / = (M−1), 一趟 $2b_r$ 个 block transfer

③ cost: $b_r(2\lceil \log_{M-1}(b_r/M) \rceil + 1)$ t_T

$(2\lceil \frac{b_r}{M} \rceil + b_r(2\lceil \log_{M-1}(b_r/M) \rceil - 1)) t_s$

init — 每一 run 一个 seek; 每一趟 $2b_r$ seeks

查询优化

① search all plans 找最优

② 启发式规则

① e.g. 用 DP 找 join order: 一般: 只考虑 left deep, 2^n [illegible]

$P(best) = \min(S_1, S - S_1, join(S_1, S - S_1))$

时间 $O(3^n)$, 空间 $O(2^n)$, (n!, n<6)

② interesting sort order

$(r_1 \bowtie r_2) \bowtie r_3$ (用 hash join $r_1 \bowtie r_2$ 更优化; 但用 merge join $r_1 \bowtie r_2$ 结果, 使后面 merge with r_3 快)

2. 启发式: 用 [illegible] 先 select (减少 tuple), 先 project (减少 column), 先做最严格的 select/join (减少 tuple)

启发式与 cost 可结合: 启发式: nested block, aggregate; cost-based: join 顺序

optimization cost: 降低 optimize 的花费

plan caching: 已有的计划重用

Join

① nested loop join: 外层每个 r, 内层所有 s: $n_r \cdot b_s + b_r$ 个 t_T, $n_r + b_r$ 个 t_s

② block: 外层每个 block r, 内层所有 blocks: $b_r \cdot b_s + b_r$ 个 t_T, $2b_r$ 个 t_s

改进: 外层用 M−2 个 block, 内层 1 个, 1 个 output

$\lceil \frac{b_r}{M-2} \rceil \cdot b_s + b_r$ 个 t_T

$2\lceil \frac{b_r}{M-2} \rceil t_s$

③ indexed (equi-join / natural join) 内 index, 外层 r

$b_r(t_T + t_s) + n_r \cdot c$ (c: 一次搜索时间)

④ merge join (已排好序的 R, S) (only equi-/natural) $b_r + b_s$ 个 t_T

while 找到一个 t_s, 找所有 t_s, 找所有 t_r = t_s

$\lceil \frac{b_r}{b_b} \rceil + \lceil \frac{b_s}{b_b} \rceil$ t_s

hybrid: (一个有序, 一个有 B+ 树): 将 sorted 与 leaf entries 合并, 之后 ... unsorted 的 address 换成 tuples 后 merge

⑤ hash join

① recursive: 内存太小: M ways 递归 (不同 hash)

② 对每个 partition 用 hash 建 hash index, 外层 r 逐个查找

划分 build: $3(b_r+b_s) + 4n_h$ 个 t_T

cost: $2(\lceil \frac{b_r}{b_b} \rceil + \lceil \frac{b_s}{b_b} \rceil)$ t_s

recursive: $2(b_r+b_s)\lceil \log_{M-1}(b_s) - 1 \rceil + b_r + b_s$ 个 t_T

$2(\lceil \frac{b_r}{b_b} \rceil + \lceil \frac{b_s}{b_b} \rceil)\lceil \log_{M-1}(b_s) - 1 \rceil$

overflow: 某 partition 过大: ① resolution: 再 partition ② avoidance

若全 hash 表无法放入 block 用 nested loop

cost estimation

n_r: tuple 个数, b_r: block 个数, l_r: size

f_r: 每个 block 的 tuple, $V(A, r)$: distinct

① selection: $\sigma_{A=v}(r)$: $\frac{n_r}{V(A,r)}$

$\sigma_{A \le v}(r) = n_r \cdot \frac{v - \min(A, r)}{\max - \min}$

条件满足率: $\frac{s_i}{n_r}$

conjunction: $n_r \cdot \frac{s_1 \times s_2 \dots \times s_n}{n_r^n}$

disjunction: $n_r(1 - (1 - \frac{s_1}{n_r}) \dots (1 - \frac{s_n}{n_r}))$

negation: $n_r - \dots$

② join: $R \cap S = \emptyset$: $r \times s$

$R \cap S$ 为 R 的 key: ≤ n_s

$R \cap S$ 为 S 的外键: $= n_r$

(4) 一般: $\frac{n_r n_s}{V(A,s)}$, $\frac{n_r n_s}{V(A,r)}$ 取小

outer join: 等值 + r / + s

distinct: $V(A, \sigma_\theta(r))$

① θ 为 $A = v$: 1

② θ 为 A 一个集合: 集合大小

③ θ 为 A op r: $V(A,r) \cdot s$

④ 一般: $\min(V(A,r), n_{\sigma\theta(r)})$

join: $\min(V(A, r), n_{r \bowtie s})$

$V(A, r \bowtie s)$

① projection: $V(A, r)$

$\Pi_A(r)$: $V(A, r)$

aggregation: min, max 等 直接取 $V(A, r)$

∩: min, ∪: +, −: r

aggregation: $A g_F(r) = V(A, r)$