多级逻辑优化

Multi-level Logic Synthesis

2025-5

主讲人：孙静翎

先回顾一下上节课的内容

两级逻辑优化

Two-level logic minimization

- 精确逻辑最小化：
  - 目标是计算出一个最小覆盖。对于大型函数是困难的
  - Quine-McCluskey方法（奎因–麦克拉斯基算法）
- 启发式逻辑最小化：
  - 致力于在短时间内计算近似的最小覆盖，这通常足够快速但可能不是最优解。
  - MINI, PRESTO, ESPRESSO

启发式最小化的操作

Heuristic minimization - operators

- 扩展（Expand）
  - 使蕴含项成为质蕴含项并移除被覆盖的蕴含项
- 缩减（Reduce）
  - 在保持覆盖范围不变的前提下，缩小每个蕴含项的大小
- 重构（Reshape）
  - 同时修改两个蕴含项：放大一个，缩小另一个
- 去冗余（Irredundant）
  - 移除覆盖中的冗余蕴含项

启发式最小化的操作

Heuristic minimization - operators

- MINI
  - 迭代执行 EXPAND、 REDUCE、 RESHAPE
  - MINI 只能保证最小的单个蕴含项上的非冗余覆盖
- Espresso
  - 迭代执行 EXPAND、 IRREDUNDANT、 REDUCE
  - Espresso 能保证无冗余覆盖，因为它包含了去冗余操作

扩展 – 简单实现

Expand - Naïve implementation

- 对于每个蕴含项
  - 对其中每个不为“无关项”（don’t care）的布尔文字
  - 如果可能，将其更改为“无关项”
  - 移除所有被扩展后的蕴含项覆盖的蕴含项
- 需要处理的问题：
  - 有效性检查
  - 决定扩展的顺序

有效性检查

Validity check

Espresso、 MINI

- 检查扩展后蕴含项与 OFF-SET的交集是否为空
- 该过程需要做求补操作

Presto

- 检查扩展后蕴含项是否包含于 ON-SET与 DC-SET的并集中
- 该问题可归约为递归重言式判定

启发式排序

Ordering heuristics

- 选择策略：

  1. 计算代表每列之和的向量（vector of column sums）

  2. 计算权重：每个蕴含项与该向量的内积（inner product）

  3. 根据权重的升序对蕴含项进行排序，选择最小权重的蕴含项

- 原理：

  较低的权重代表某个蕴含项在密集列中含有较少的 1（即可能有更多其他蕴含项与其相反）

不是作业

请用矩阵表示法对以下布尔函数扩展：

$$F = a'bcd + a'bcd' + a'b'cd$$

- 对于每个蕴含项
  - 对其中每个不为“无关项”（don’t care）的布尔文字
  - 如果可能，将其更改为“无关项”
  - 检查新蕴含项与F’交集是否为空，若是，移除所有被扩展后的蕴含项覆盖的蕴含项

例子

$F = a'bcd+a'bcd'+a'b'cd$

1. 求F'

对变量b取余因子

$F' = aF'_a+F'_{a'}$

$= a+F'_{ac} +c'F'_{ac'}$

$= a+F'_{ac} +c'$

$= a+b\ F'_{abc} +b'\ F'_{ab'c} +c'$

$= a+b'd'+c'$

F_{abc}: 11 01 11 01
11 01 11 10
11 10 11 01
11 01 11 11

11 01 11 01
11 01 11 10
11 00 11 01
00 10 00 00

11 11 11 01
11 11 11 10

F_{abc}=1 -> F'_{abc}=0

$F_{ab'c}$: 11 01 11 01
11 01 11 10
11 10 11 01
11 10 11 11

11 00 11 01
11 00 11 10
11 10 11 01
00 01 00 00

$F_{ab'c}$: 11 11 11 01

$F'_{ab'c}$:11 11 11 10

例子

- F = a’bcd+a’bcd’+a’b’cd

10 01 01 01

10 01 01 10

10 10 01 01

- 排序:
  - 1. 计算代表每列之和的向量： $[3\ 0\ 1\ 2\ 0\ 3\ 1\ 2]^T$
  - 2. 计算权重：每个蕴含项与该向量的内积：

    10（a’bcd），9（a’bcd’），9（a’b’cd）
  - 3. 根据权重的升序对蕴含项进行排序，选择最小权重的蕴含项：

    选择a’bcd’或a’b’cd

- F = a’bcd+a’bcd’+a’b’cd

10 01 01 01
10 01 01 10
10 10 01 01

F’= a+b’d’+c’

01 11 11 11
11 10 11 10
11 11 10 11

- 扩展a’bcd’（10 01 01 10）：
  - 扩展为bcd’（11 01 01 10） 无效.
  - 扩展为a’cd’（10 11 01 10）无效.
  - 扩展为a’bd’（10 01 11 10） 无效.
  - 扩展为a’bc（10 01 01 11） 有效.
- 最终扩展a’bcd’为a’bc， 移除所有被扩展后的蕴含项覆盖的蕴含项:
  - a’bcd’和a’bcd
- 更新后的覆盖

10 01 01 11
10 10 01 01

- F = a’bc+a’b’cd　10 01 01 11
  10 10 01 01

F’= a+b’d’+c’　01 11 11 11
11 10 11 10
11 11 10 11

- 扩展a’b’cd （ 10 10 01 01）：
  - 扩展为b’cd（11 10 01 01） 无效.
  - 扩展为a’cd（10 11 01 01）有效.
  - 扩展为a’d（10 11 11 01） 无效.
  - 扩展为a’c（10 11 01 11）无效.
- 最终扩展a’b’cd 为a’cd， 移除所有被扩展后的蕴含项覆盖的蕴含项:
  - a’b’cd和a’bcd
- 更新后的覆盖

10 01 01 11
10 11 01 01

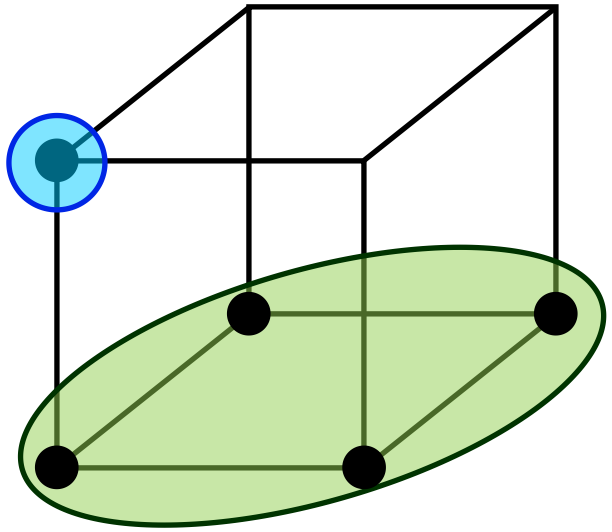
缩减 – 简单实现

Reduce - Naïve implementation

- 对于每个蕴含项
  - 对其中每个无关项”（don’t care）
  - 如果可能，将其更改为布尔文字1或0
  - 有效性检查：要求新的覆盖的补和原蕴含项交集为空集，或者新的覆盖包含原蕴含项

例子

- F=c’+a’b’ **11 11 10**
10 10 11
- 选择第一个蕴含项：
  - 尝试：11 10 10, 11 01 10, 10 11 10, 01 11 10，都不行
  - 无法进行约简。
- 选择第二个蕴含项：
  - 尝试：10 10 10, 10 10 01，发现第二个可以
  - 约简为 10 10 01
- 约简后的覆盖：**11 11 10**
10 10 01



缩减

Reduce

- 最优缩减可以被精确地确定
- 定理：
  - 当要对蕴含项α进行缩减时
  - Q=除α以外的剩余的蕴含项相加
  - 则α缩减后的蕴含项为：

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q\text{'对}\alpha\text{取余因子})$$

不是作业

请用矩阵表示法对以下布尔函数缩减：

$$F = a'bc + a'cd$$

- 对于每个蕴含项
  - Q=除α以外的剩余的蕴含项相加
  - 则α缩减后的蕴含项为：

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\text{对}\alpha\text{取余因子})$$

例子

- F=a’bc+a’cd **10 01 01 11**
  10 11 01 01
- 如果先选择第一个蕴含项(a’bc)作为α:

  á = α ∩ supercube(Q’对α取余因子)
  - Q=除α以外的剩余的蕴含项相加=a’cd
  - Q’=a+c’+d’
  - Q’α: d’
  - supercube(Q’α)=d’
  - á = α ∩ supercube(Q’α)=a’bcd’ (最优化简)
  - 缩减后的覆盖: **10 01 01 10**
    10 11 01 01

01 11 11 11

11 11 10 11

11 11 11 10

10 01 01 11

---

00 01 01 11

10 01 00 11

10 01 01 10

01 10 10 00

---

11 11 11 10

例子

- F=a’bcd’+a’cd **10 01 01 10**
  10 11 01 01
- 再选择第二个蕴含项(a’cd)作为α:

  á = α ∩ supercube(Q’对α取余因子)
  - Q=除α以外的剩余的蕴含项相加=a’bcd’
  - Q’=a+b’+c’+d
  - Q’α:1
  - supercube(Q’α)=1
  - á = α ∩ supercube(Q’α)=a’cd (无法化简)
  - 最终F= a’bcd’+a’cd

01 11 11 11
11 10 11 11
11 11 10 11
11 11 11 01
10 11 01 01

---

00 11 01 01
10 10 01 01
10 11 00 01
10 11 01 01
01 00 10 10

---

11 10 11 11
11 11 11 11

不是作业

请用矩阵表示法对以下布尔函数缩减：

$$F = a'bc + a'cd$$

- 对于每个蕴含项
  - Q=除α以外的剩余的蕴含项相加
  - 则α缩减后的蕴含项为：

$$\acute{\alpha} = \alpha \cap \text{supercube}(Q'\text{对}\alpha\text{取余因子})$$

例子-第二种做法

- F=a’bc+a’cd　　**10 01 01 11**
  10 11 01 01
- 如果先选择第二个蕴含项(a’cd)作为α:

  á = α ∩ supercube(Q’对α取余因子)
  - Q=除α以外的剩余的蕴含项相加=a’bc
  - Q’=a+b’+c’
  - Q’α: b’
  - supercube(Q’α)=b’
  - á = α ∩ supercube(Q’α)=a’b’cd (最优化简)
  - 缩减后的覆盖:　　**10 01 01 11**
    10 10 01 01

01 11 11 11
11 10 11 11
11 11 10 11
10 11 01 01

---

00 11 01 01
10 10 01 01
10 11 00 01
01 00 10 10

---

11 10 11 11

例子-第二种做法

- F=a’bc+a’b’cd **10 01 01 11**
  10 10 01 01
- 选择第一个蕴含项(a’bc)作为α：

  á = α ∩ supercube(Q’对α取余因子)

  - Q=除α以外的剩余的蕴含项相加=a’b’cd
  - Q’=a+b+c’+d’
  - Q’α: 1
  - supercube(Q’α)=1
  - á = α ∩ supercube(Q’α)=a’bc (无法化简)
  - 最终F= a’bc+a’b’cd

01 11 11 11
11 01 11 11
11 11 10 11
11 11 11 10
10 01 01 11

---

00 01 01 11
10 01 01 11
10 01 00 11
10 01 01 10
01 10 10 00

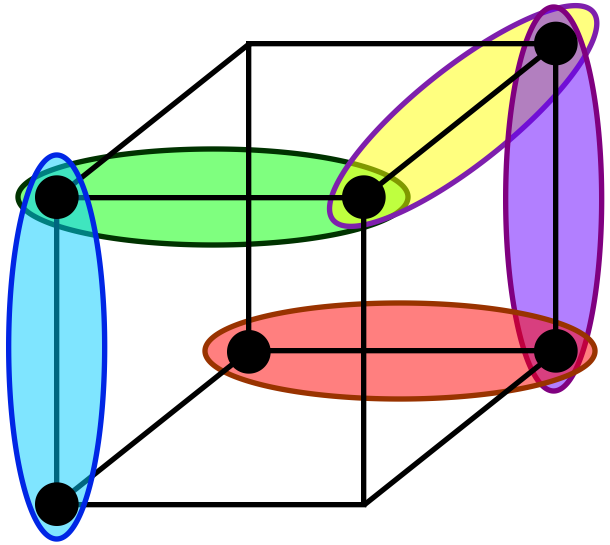
---

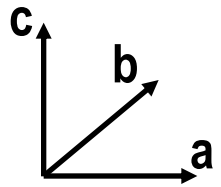
11 11 11 11
11 11 11 10

去冗余

Irredundant

α	10	10	11
β	11	10	01
γ	01	11	01
δ	01	01	11
ε	11	01	10





去冗余

Irredundant

- 相对必要集（Er）
  - 覆盖了函数中某些未被其他蕴含项覆盖的最小项的蕴含项组成的集合。
  - 重要说明：做启发式逻辑优化时我们尚未知道所有的质蕴含项，只是相对必要
- 完全冗余集（Rt）
  - 被相对必要项所覆盖的蕴含项集合。
- 部分冗余集（Rp）
  - 剩余的蕴含项集合。

去冗余

Irredundant

- 相对必要集（Er）
  - Q = 除某个蕴含项α以外的剩余的蕴含项相加
  - 若Q不包含α，则α是相对必要项 <=> Q_{α}不为重言式
- 完全冗余集（Rt）
  - 若Er包含某个蕴含项α，则α是完全冗余项
- 部分冗余集（Rp）
  - Rp = F - (Er U Rt)

去冗余

Irredundant

- 寻找一个 Rp(部分冗余集) 的子集，使其与 Er (相对必要集)一同覆盖整个函数
- 对Rp中的每个蕴含项α，判断其是否被H=Rp+Er-{α}这个集合组成的覆盖所包含，若包含，则可以从Rp中去掉该蕴含项
- 排除所有可以去掉的蕴含项，剩余的蕴含项与Er组成非冗余覆盖

作业

请用矩阵表示法对以下布尔函数去冗余：

$$F = a'b + a'd' + b'd' + ab'$$

- 相对必要集（Er）
  - 判断每个蕴含项α是否是相对必要集：
    - Q = 除某个蕴含项α以外的剩余的蕴含项相加
    - 若Q不包含α，则α是相对必要项
- 完全冗余集（Rt）
  - 若Er包含某个蕴含项α，则α是完全冗余项
- 部分冗余集（Rp）
  - Rp = F - (Er U Rt)

- 对Rp中的每个蕴含项α，判断其是否被H=Rp+Er-{α}这个集合组成的覆盖所包含，若包含，则可以从Rp中去掉该蕴含项
- 排除所有可以去掉的蕴含项，剩余的蕴含项与Er组成非冗余覆盖

例子

F= a’b+a’d’+b’d’+ab’

判断a’b是否是相对必要项

Q= a’d’+b’d’+ab’

$Q_{a'b}$不为重言式=> Q不包含a’b => a’b是相对必要项

Er = {a’b}

10 11 10
11 10 10
01 10 11
10 01 11

---

10 01 10
10 00 10
00 00 11
01 10 00

---

11 11 10

例子

F= a’b+a’d’+b’d’+ab’

判断a’d’是否是相对必要项

Q= a’b+b’d’+ab’

$Q_{a'd'}$为重言式=>Q包含a’d’ =>a’d’不是相对必要项

Er = {a’b}

10 01 11
11 10 10
01 10 11
10 11 10

---

10 01 10
10 10 10
00 10 10
01 00 01

---

11 01 11
11 10 11

例子

F= a’b+a’d’+b’d’+ab’

判断b’d’是否是相对必要项

Q= a’b+a’d’+ab’

$Q_{b'd'}$为重言式=>Q包含b’d’ =>b’d’不是相对必要项

Er = {a’b}

10 01 11
10 11 10
01 10 11
11 10 10

---

10 00 10
10 10 10
01 10 10
00 01 01

---

10 11 11
01 11 11

例子

F= a’b+a’d’+b’d’+ab’

判断ab’是否是相对必要项

Q= a’b+a’d’+b’d’

$Q_{ab'}$不为重言式=>Q不包含ab’ => ab’是相对必要项

Er = {a’b, ab’}

10 01 11
10 11 10
11 10 10
01 10 11

---

00 00 11
00 10 10
01 10 10
10 01 00

---

11 11 10

例子

F= a’b+a’d’+b’d’+ab’

Er = {a’b, ab’}

判断a’d’是否是完全冗余项

Q=a’b+ab’，通过判断$Q_{a'd'}$是否是重言式判断包含

Er不包含a’d’=>a’d’不是完全冗余项

Rp = {a’d’}

10 01 11
01 10 11
10 11 10
---
10 01 10
00 10 10
01 00 01
---
11 01 11

例子

F= a’b+a’d’+b’d’+ab’

Er = {a’b, ab’}

判断b’d’是否是完全冗余项

Q=a’b+ab’，通过判断$Q_{b'd'}$是否是重言式判断包含

Er不包含b’d’=>b’d’不是完全冗余项

最终得出:

Er = {a’b, ab’}

Rt = ∅

Rp = {a’d’, b’d’}

10 01 11
01 10 11
11 10 10
---
10 00 10
01 10 10
00 01 01
---
01 11 11

例子

F= a’b+a’d’+b’d’+ab’

Er = {a’b, ab’}

Rt = ∅

Rp = {a’d’, b’d’}

判断a’d’是否可以被去掉

H=a’b+ab’+b’d’，通过判断$H_{a'd'}$是否是重言式判断包含

$H_{a'd'}$是重言式=>H包含a’d’=>a’d’可以被去掉

10 01 11
01 10 11
11 10 10
10 11 10

---

10 01 10
00 10 10
10 10 10
01 00 01

---

11 01 11
11 10 11

例子

F= a’b+a’d’+b’d’+ab’

Er = {a’b, ab’}

Rt = ∅

Rp = {b’d’}

10 01 11
01 10 11
11 10 10
---
10 00 10
01 10 10
00 01 01
---
01 11 11

判断b’d’是否可以被去掉

H=a’b+ab’，通过判断$H_{b'd'}$是否是重言式判断包含

$H_{b'd'}$不是重言式=>H不包含b’d’=>b’d’不可以被去掉

最终非冗余覆盖：{a’b, ab’, b’d’}

启发式最小化的操作

Heuristic minimization - operators

- MINI
  - 迭代执行 EXPAND、 REDUCE、 RESHAPE
  - MINI 只能保证最小的单个蕴含项上的非冗余覆盖
- Espresso
  - 迭代执行 EXPAND、 IRREDUNDANT、 REDUCE
  - Espresso 能保证无冗余覆盖，因为它包含了去冗余操作

ESPRESSO算法

ESPRESSO algorithm

```
espresso(F) {
    repeat {
        φ1 = |F |;
        F = reduce(F);
        F = expand(F);
        F = irredundant(F);
    } until (|F | ≥ φ1);
}
```

ESPRESSO算法

ESPRESSO algorithm

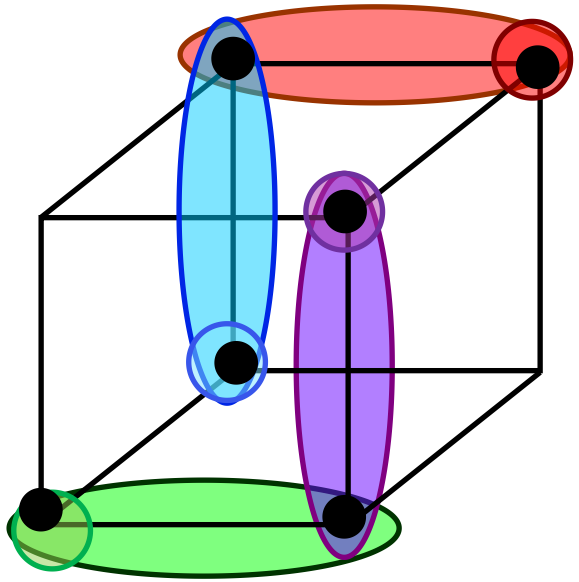
- 化简布尔函数（只保留部分冗余集）
- 迭代执行：
  - 扩展（Expand）、去冗余（Irredundant）和化简（Reduce）
- 代价函数（Cost Functions）：
  - φ1：蕴含项数
  - φ2：蕴含项数与布尔文字数的加权和

```
espresso(F, D) {
    R = 计算 (F ∪ D) 的补集;
    F = 扩展(F, R);
    F = 去冗余(F, D);
    E = 提取相对必要集;
    F = F - E;   D = D ∪ E;
    重复执行 {
        φ2   = 当前 F 的蕴含项和布尔文字加权成本;
        重复执行 {
            φ1   = 当前 F 的蕴含项数量;
            F = 化简(F, D);
            F = 扩展(F, R);
            F = 去冗余(F, D);
        } 直到 (|F| ≥ φ1   );
        F = last_gasp(F, D, R);
    } 直到 (cost(F) ≥ φ2   );
    F = F ∪ E;
    D = D - E;
    F = make_sparse(F, D, R);
}
```

last_gasp函数

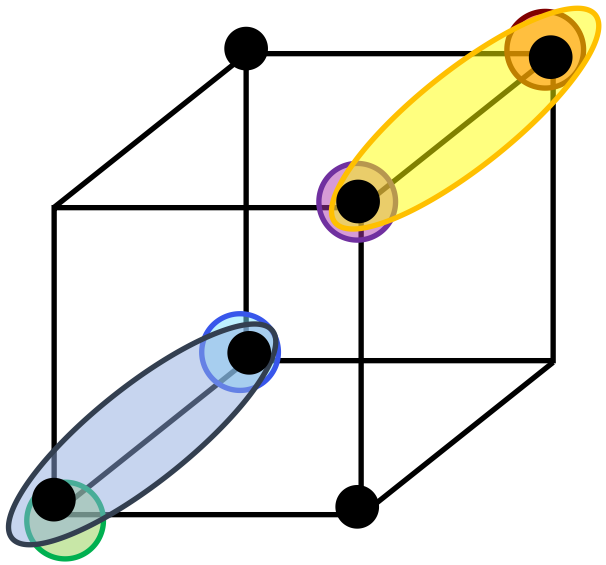
1.独立地对各个蕴含项进行约简（约简顺序无关）

注意：所得的约简蕴含项集合 {c1, c2, ... cp} 不一定构成一个覆盖。



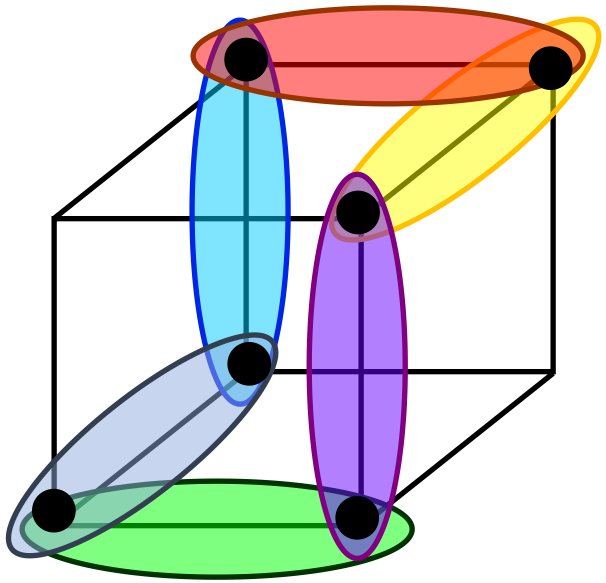
last_gasp函数

2. 对约简后的立方项进行扩展，生成一组新的质蕴含项（NEW_PR）



last_gasp函数

3. 使用 NEW_PR ∪ OLD_PR 作为输入，运行 IRREDUNDANT（去冗余操作），其中 OLD_PR 是原有的质蕴含项集合

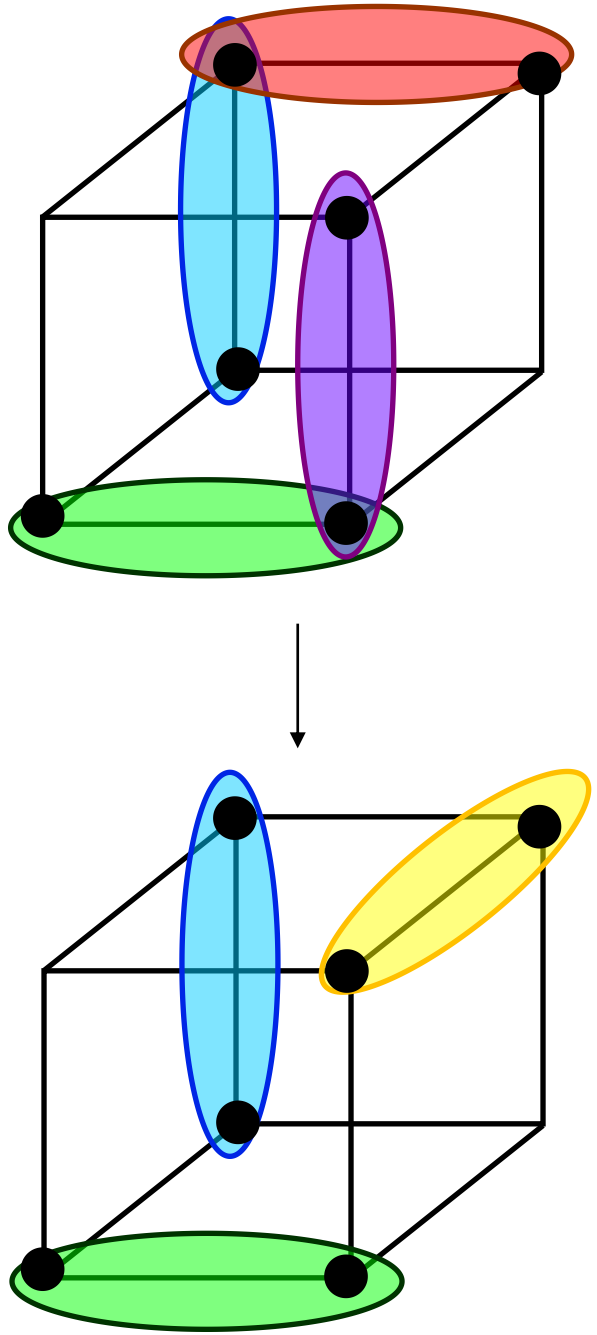


last_gasp函数

1.独立地对各个蕴含项进行约简

2. 对约简后的立方项进行扩展，生成一组新的质蕴含项（NEW_PR）

3. 使用 NEW_PR ∪ OLD_PR 作为输入，运行 IRREDUNDANT（去冗余操作），其中 OLD_PR 是原有的质蕴含项集合



```
espresso(F, D) {
    R = 计算 (F ∪ D) 的补集;
    F = 扩展(F, R);
    F = 去冗余(F, D);
    E = 提取相对必要集;
    F = F - E;  D = D ∪ E;  //至此，F中剩下的都是部分冗余集
    重复执行 {
        φ2  = 当前 F 的代价函数（加权成本）;
        重复执行 {
            φ1  = 当前 F 的大小（蕴含项数量）;
            F = 化简(F, D);
            F = 扩展(F, R);
            F = 去冗余(F, D);
        } 直到 (F 的大小不再减少，即 |F| ≥ φ1  );
        F = last_gasp(F, D, R);
    } 直到 (F 的代价函数不再降低，即 cost(F) ≥ φ2  );
    F = F ∪ E;  // 将必要项合并回结果中
    D = D - E;  // 从 Don't Care 集中移除必要项
    F = make_sparse(F, D, R);  //在不改变蕴含项数量的前提下，调整布尔文字的数量
}
```

启发式两级逻辑优化总结

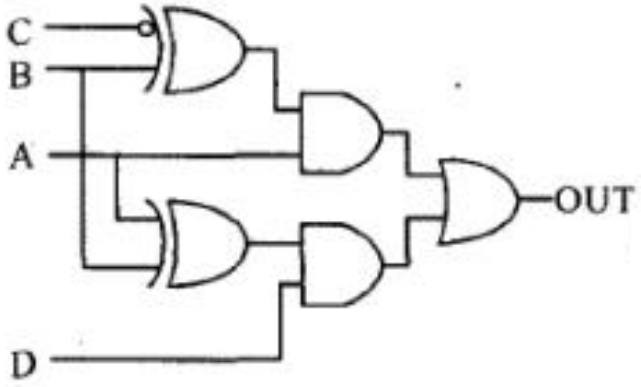
Heuristic two-level minimization Summary

- 启发式最小化是迭代式的
- 只对覆盖集应用少量操作（重言式判断，包含判断，取补操作）
- 基础机制包括：
  - 基于矩阵表示法运算
  - 基于单调性的启发算法
- 高效算法

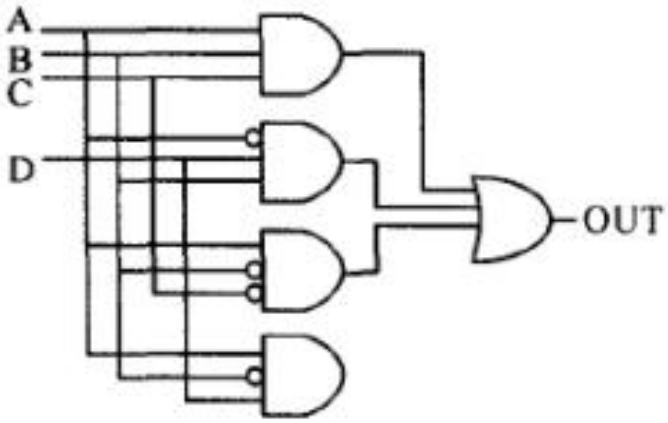
正式开始本周的内容

多级逻辑优化

Multi-level logic minimization



多级逻辑



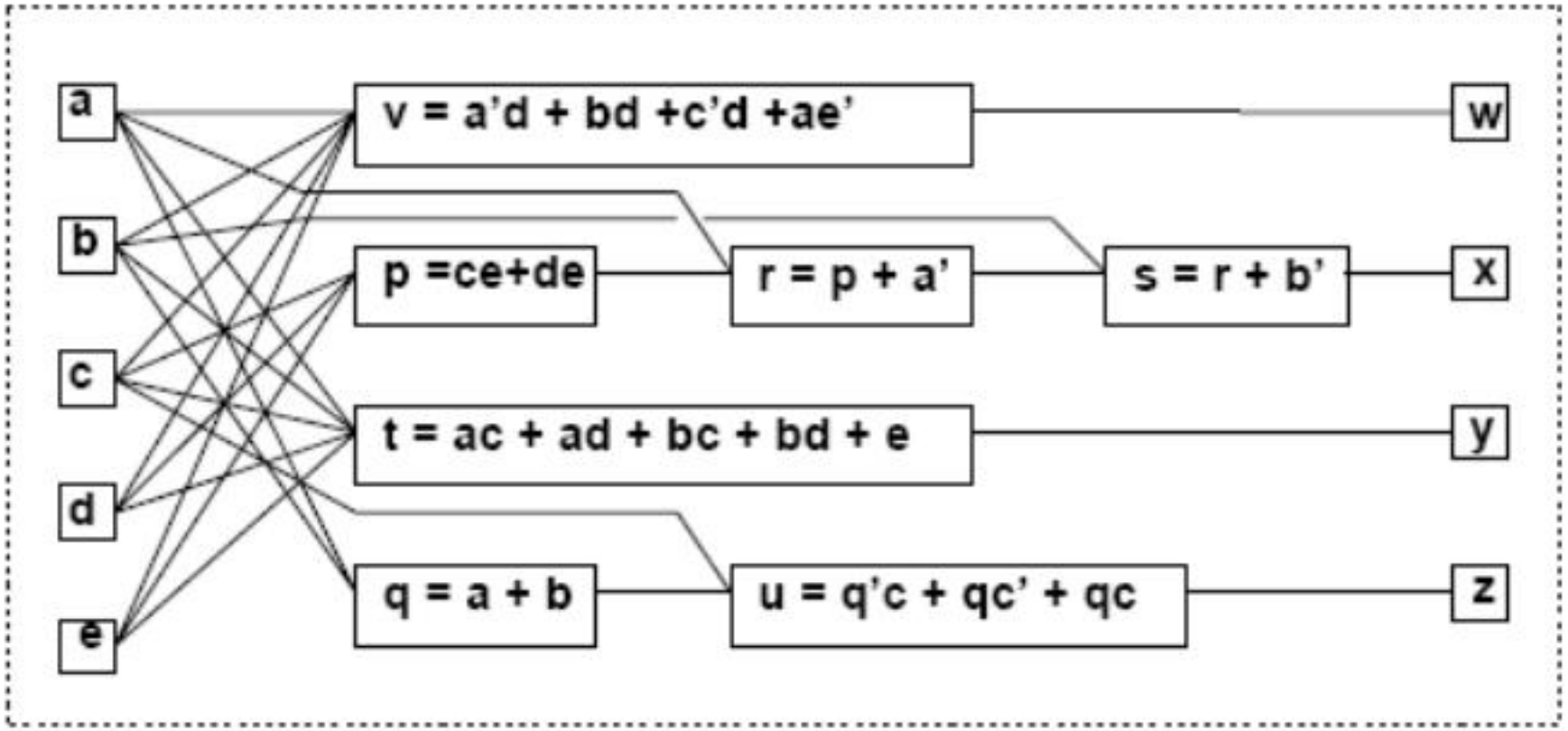
两级逻辑

多级逻辑

Multi-level logic

- 逻辑网络
  - 由多个模块互连而成的结构
  - 每个模块由一个布尔函数建模

一个多级逻辑网络的例子

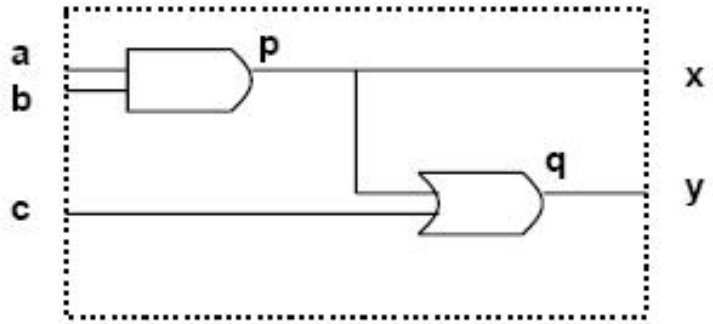


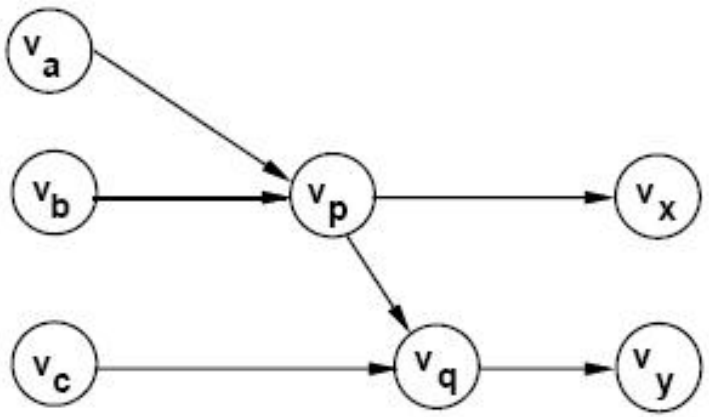
多级逻辑

Multi-level logic

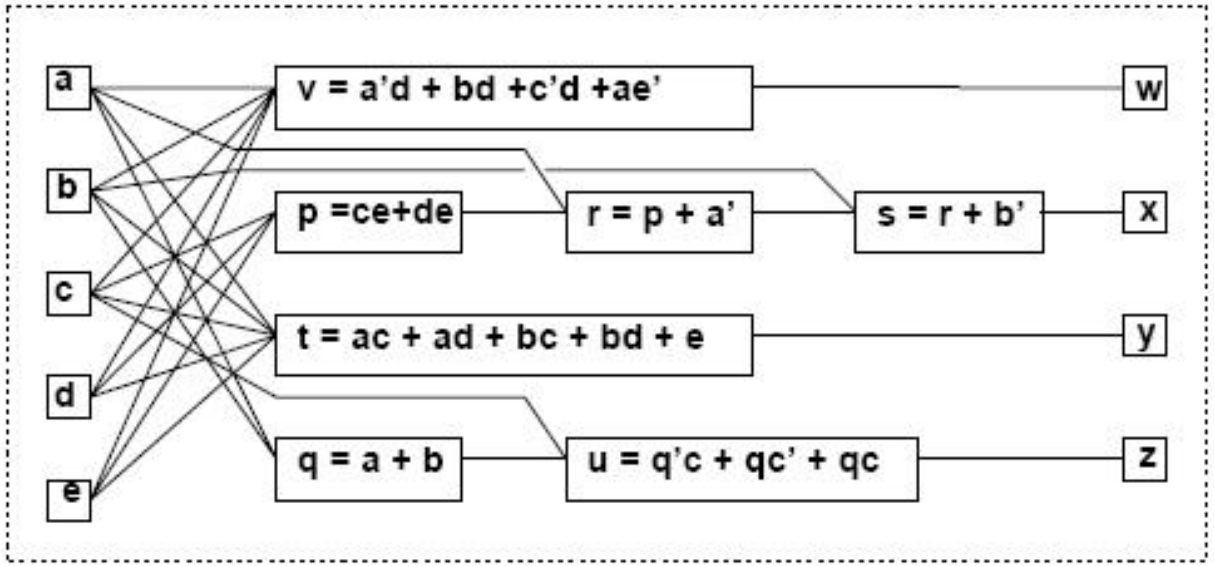
- 逻辑网络
  - 由多个模块互连而成的结构
  - 每个模块由一个布尔函数建模
- 常见的限制条件：
  - 无环
  - 每个函数仅有一个输出、
- 结构由模块之间的连接关系决定

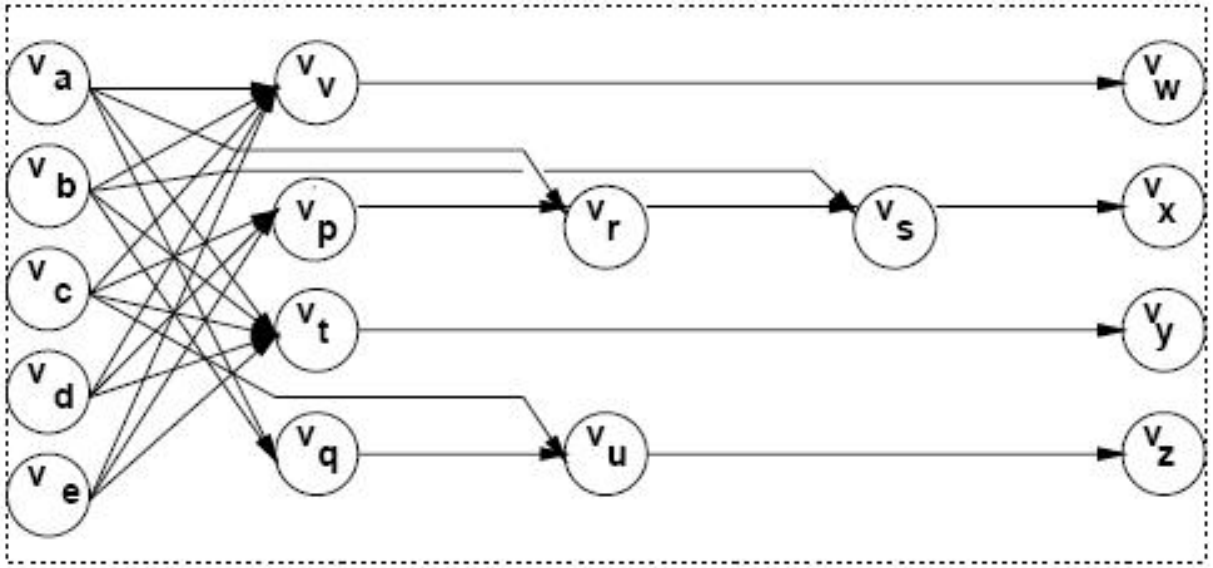
用图建模网络结构



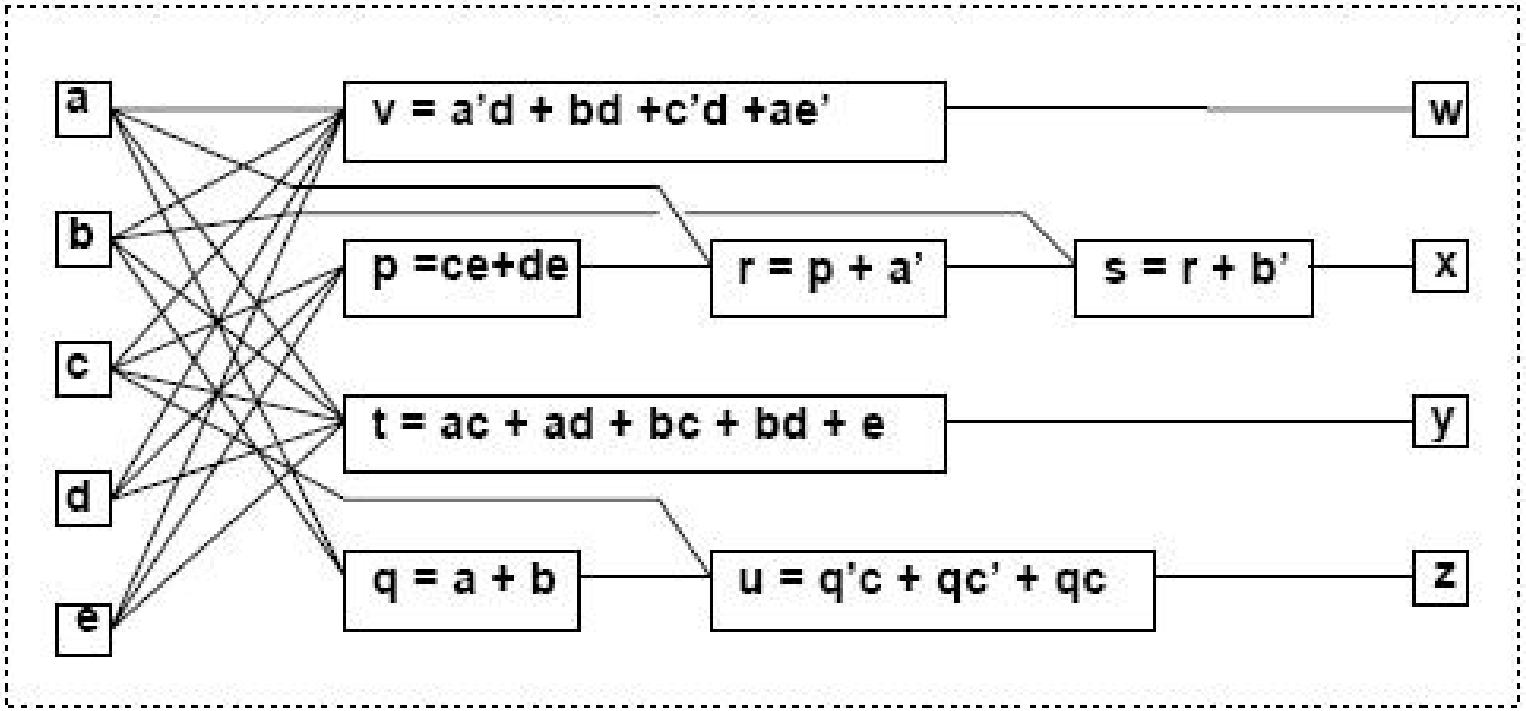


该多级逻辑网络对应的图





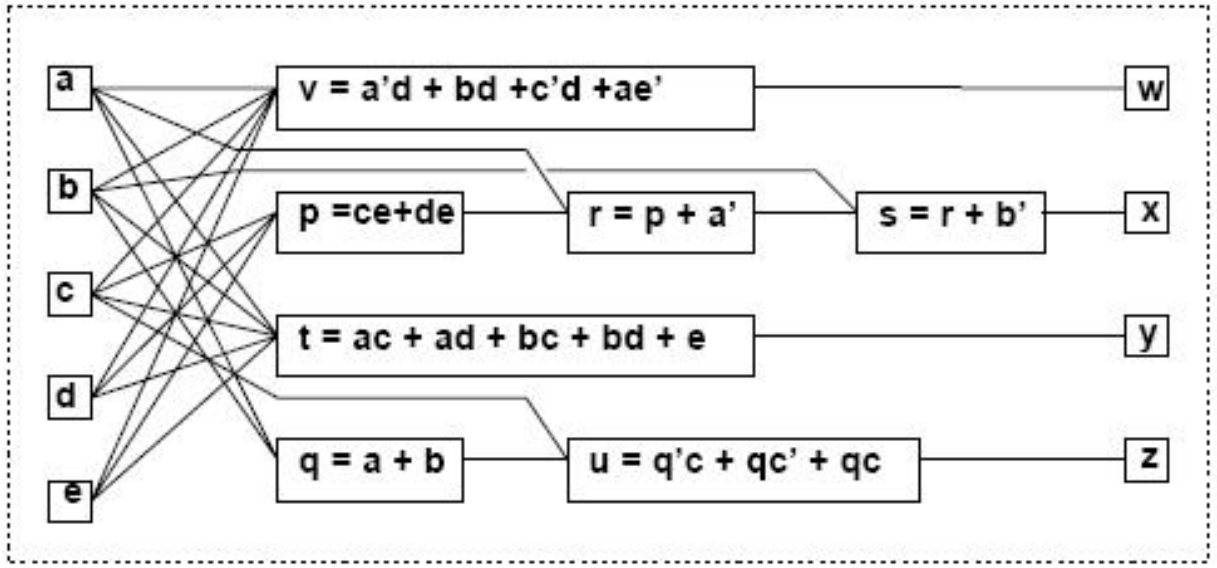
该多级逻辑网络结点的表达式表示



$$
\begin{aligned}
p &= ce + de \\
q &= a + b \\
r &= p + a' \\
s &= r + b' \\
t &= ac + ad + bc + bd + e \\
u &= q'c + qc' + qc \\
v &= a'd + bd + c'd + ae' \\
w &= v \\
x &= s \\
y &= t \\
z &= u
\end{aligned}
$$

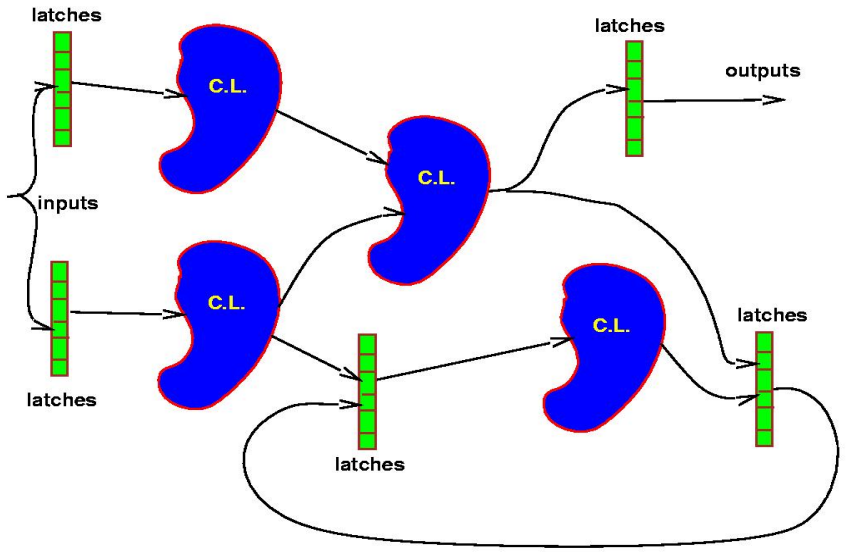
该多级逻辑网络的最终行为

$$f = \begin{bmatrix} a'd + bd + c'd + ae' \\ a' + b' + ce + de \\ ac + ad + bc + bd + e \\ a + b + c \end{bmatrix}$$



多级逻辑优化

Multi-level logic minimization



- 多级逻辑网络
  - 提供更多灵活性
  - 可在特定路径上基于不同的约束优化
  - 提升性能表现
- 逻辑综合的重要性随着面向半定制市场的代工厂的发展而同步提升

优化目标与约束

- 最小化最大延迟

  （在面积或功耗受限的条件下）

- 最小化面积

  （在延迟受限的条件下）

- 最小化功耗

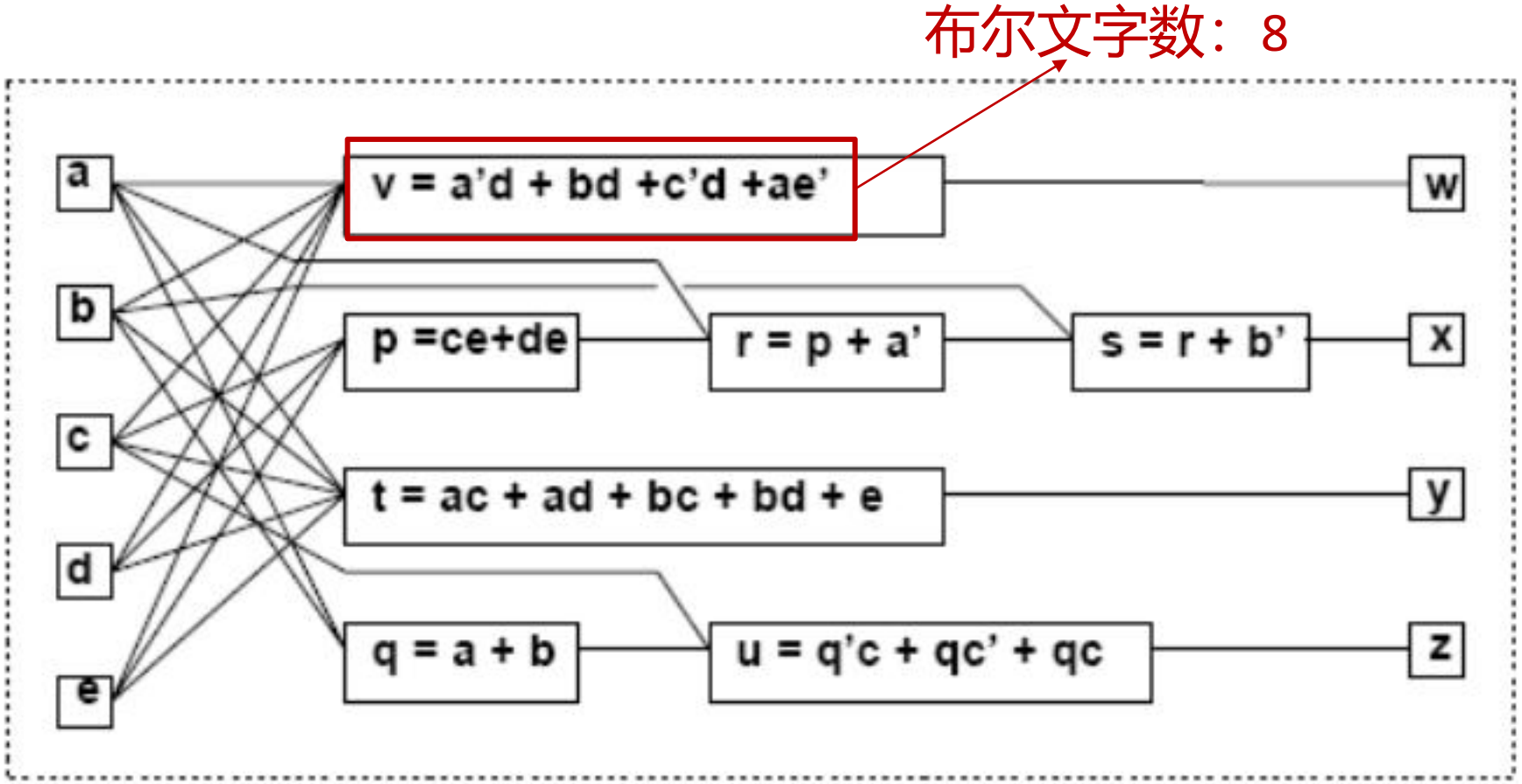
  （在延迟受限的条件下）

估算指标

Estimation

- 面积（Area）：
  - 使用布尔文字（literals）的数量来估算
    - 计算简单
    - 被广泛接受
    - 是一种良好的面积估算指标
- 延迟（Delay）：
  - 标准元件可以根据库文件直接得到
  - 非标准元件可以：
    - 根据级数（stages）粗略估计
    - 或根据逻辑函数的复杂度和扇出估算
- 功耗（Power）：
  - 功耗模型通常需要对电路所处的工作环境做出一定的假设
    - 每个节点的切换活动（switching activity）
    - 电容负载（capacitive loads）

一个多级逻辑网络的例子



估算指标

Estimation

- 面积（Area）：
  - 使用布尔文字（literals）的数量来估算
    - 计算简单
    - 被广泛接受
    - 是一种良好的面积估算指标
- 延迟（Delay）：
  - 标准元件可以根据库文件直接得到
  - 非标准元件可以：
    - 根据级数（stages）粗略估计
    - 或根据逻辑函数的复杂度和扇出估算
- 功耗（Power）：
  - 功耗模型通常需要对电路所处的工作环境做出一定的假设
    - 每个节点的切换活动（switching activity）
    - 电容负载（capacitive loads）

问题分析

- 即使是最简单的问题，其计算本质也是困难的。
  - 例如，多输入单输出的网络问题
- 目前只提出了少数精确算法：
  - 计算复杂度很高
  - 仅适用于小规模电路
- 近似优化方法：
  - 基于规则的方法（Rule-based methods）
  - 启发式算法（Heuristic algorithms）

优化策略

逐步改进逻辑网络

- 通过电路变换来优化
- 保持网络的输入输出行为不变
- 如有需要，可利用环境中的“无关项” （don’t care 条件）

不同方法的差异体现在：

- 所采用的变换类型
- 变换的选择方式和应用顺序

变换方式总结

1.消除

- 执行变量替换操作

2.拆分

- 将一个函数拆分为更小的函数

3.提取

- 在两个（或多个）表达式中查找公共子表达式

4.替代

- 通过引入一个原本不属于原输入集的已有布尔函数来简化局部函数

5.化简

- 两级逻辑优化

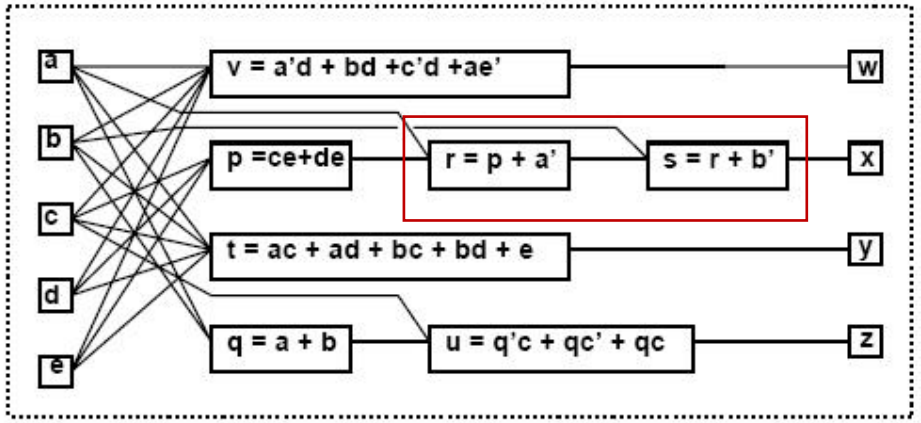
消除

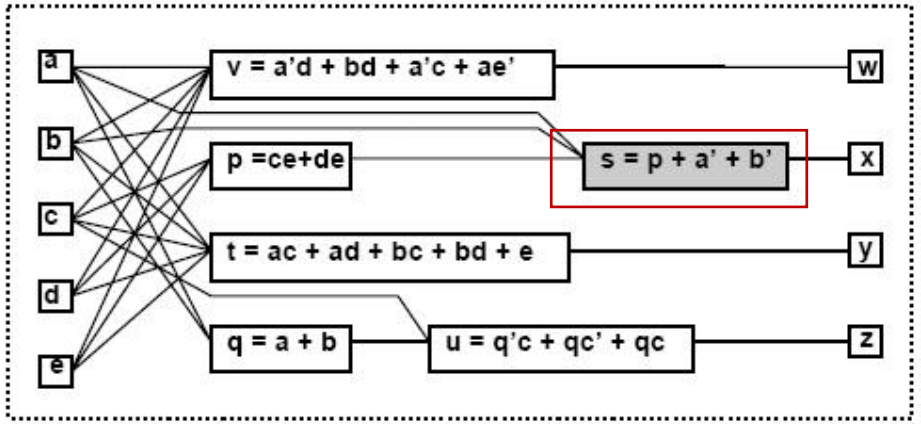
Elimination

- 从网络中消除一个函数
  - 类似于高斯消元
- 执行变量替换操作

- Example:
  - s = r + b’; r = p + a’;
  - s = p + a’ + b’;

例子


a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z


a
b
c
d
e
v = a'd + bd + a'c + ae'
p =ce+de
s = p + a' + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

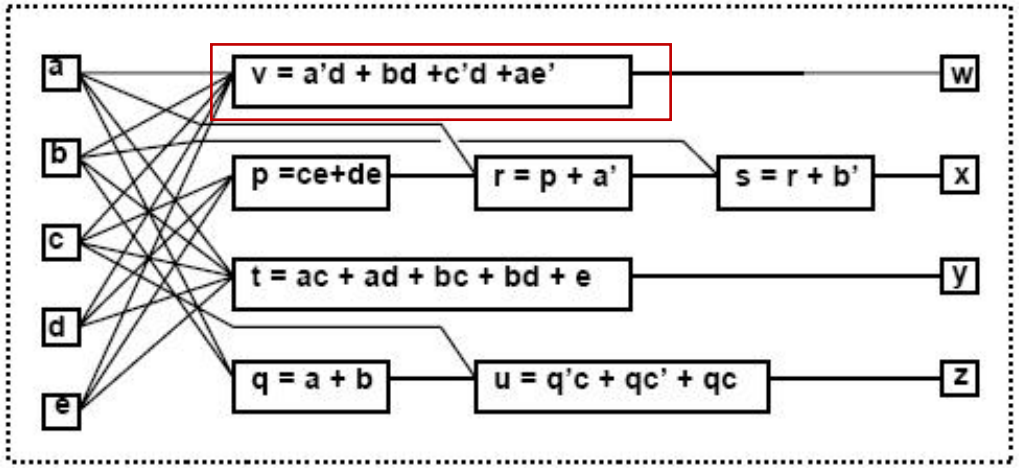
拆分

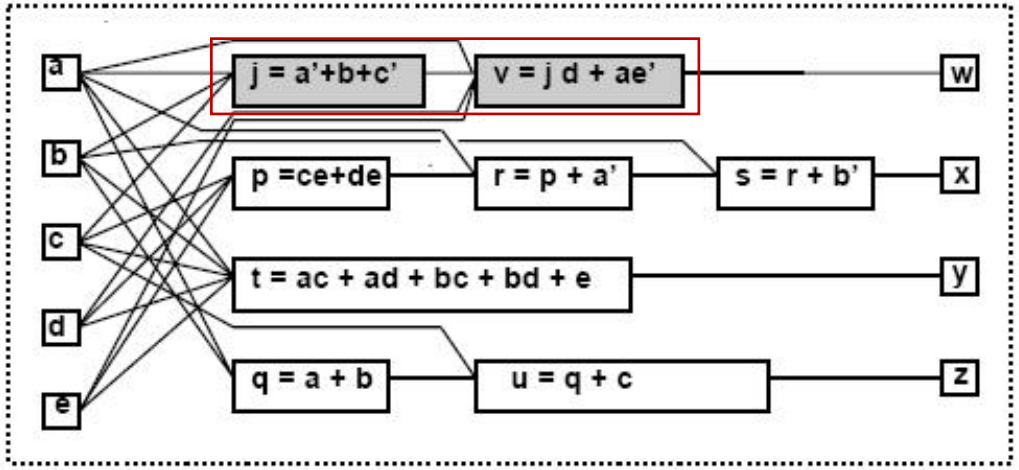
Decomposition

- 将一个函数拆分为更小的函数
  - 与“消除”操作相反
- 向网络中引入新的变量或模块

- Example:
  - v = a’d + bd +c’d +ae’
  - j = a’ + b + c’; v = jd + ae’;

例子


a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z


a
b
c
d
e
j = a'+b+c'
v = j d + ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q + c
w
x
y
z

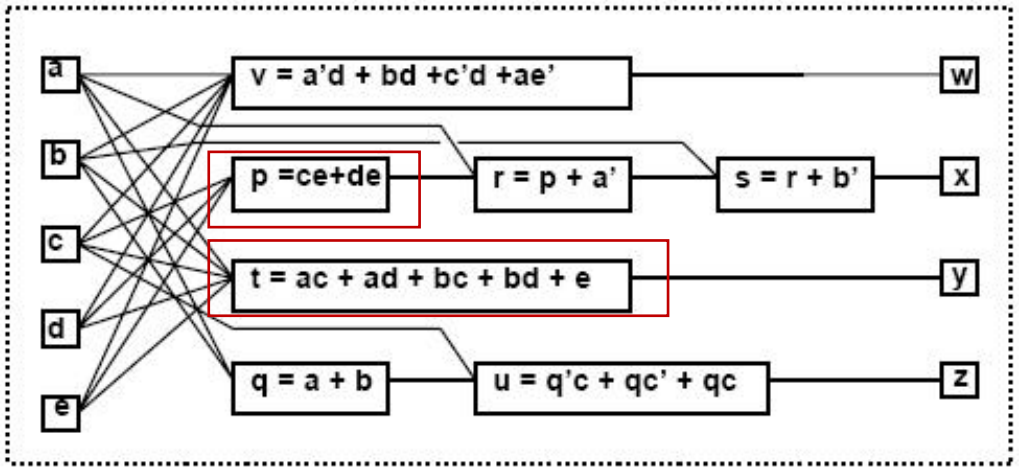
提取

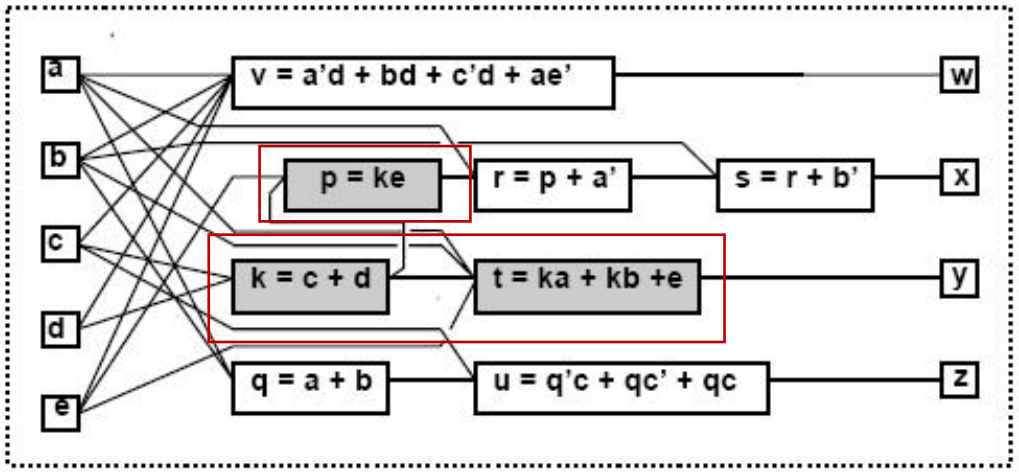
Extraction

- 在两个（或多个）表达式中查找公共子表达式
  - 提取该子表达式作为新的函数
  - 向电路中引入一个新的模块

- Example
  - p = ce + de; t = ac + ad + bc + bd + e;
  - p = (c + d) e; t = (c + d) (a + b) + e;
  - k = c + d; p = ke; t = ka + kb + e;

例子


a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z


a
b
c
d
e
v = a'd + bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = ka + kb +e
q = a + b
u = q'c + qc' + qc
w
x
y
z

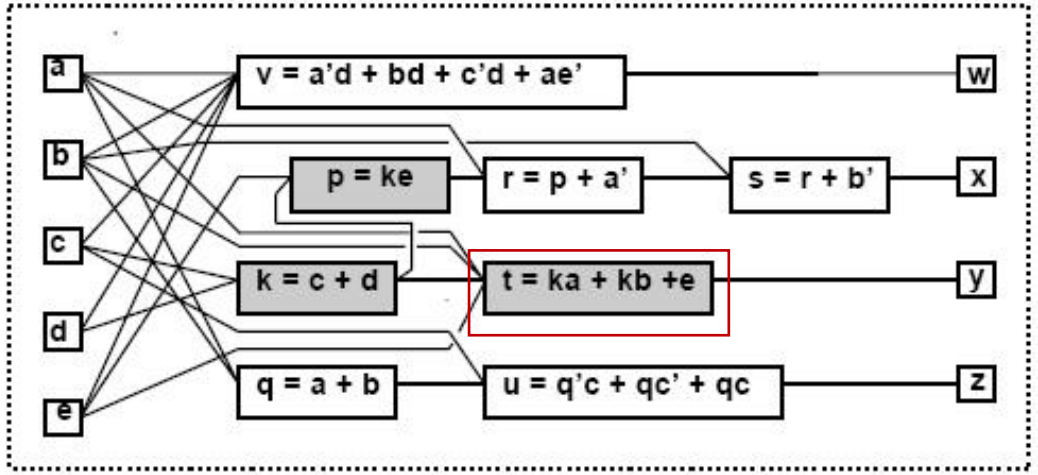
替代

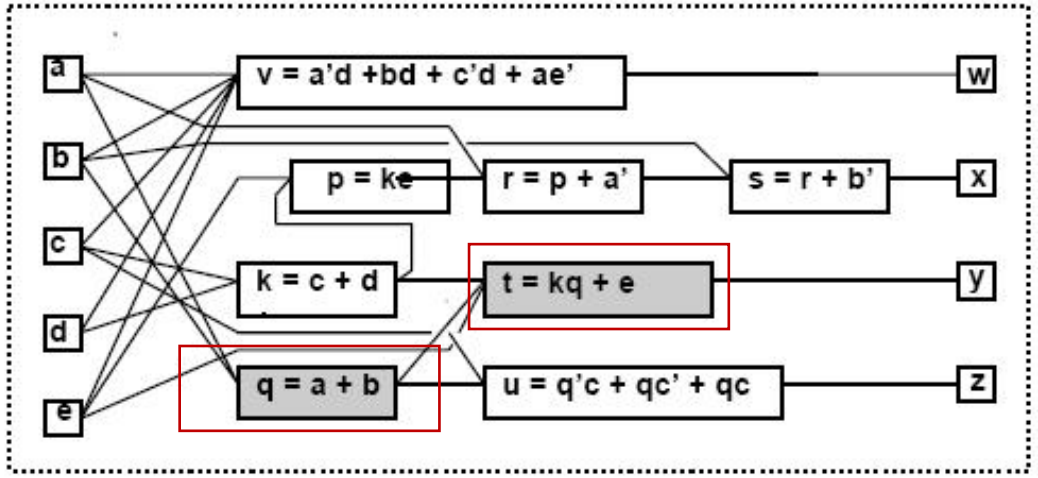
Substitution

- 通过引入一个原本不属于原输入集的已有布尔函数来简化局部函数

- Example:
  - t = ka + kb + e;
  - t = kq + e;
  - Because q = a + b is already part of the network

例子


a
b
c
d
e
v = a'd + bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = ka + kb +e
q = a + b
u = q'c + qc' + qc
w
x
y
z

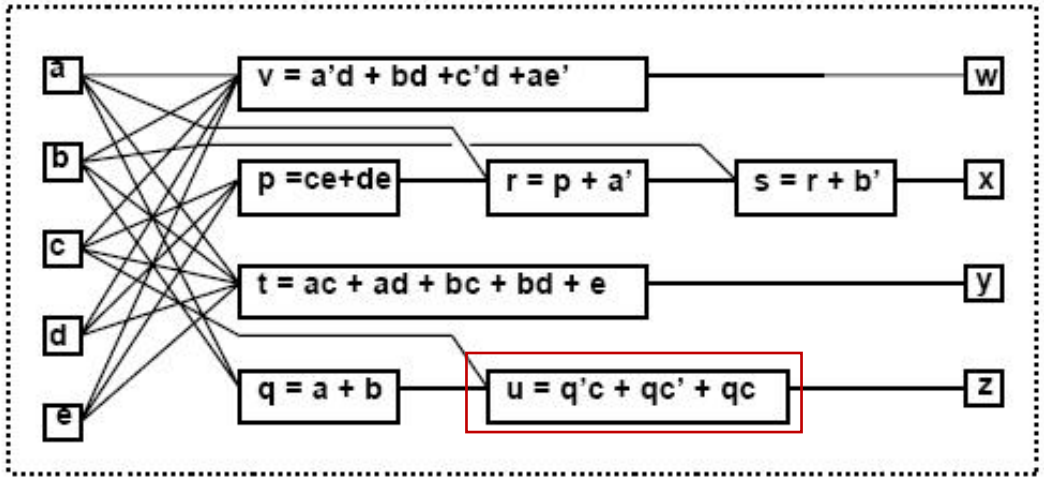

a
b
c
d
e
v = a'd +bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = kq + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

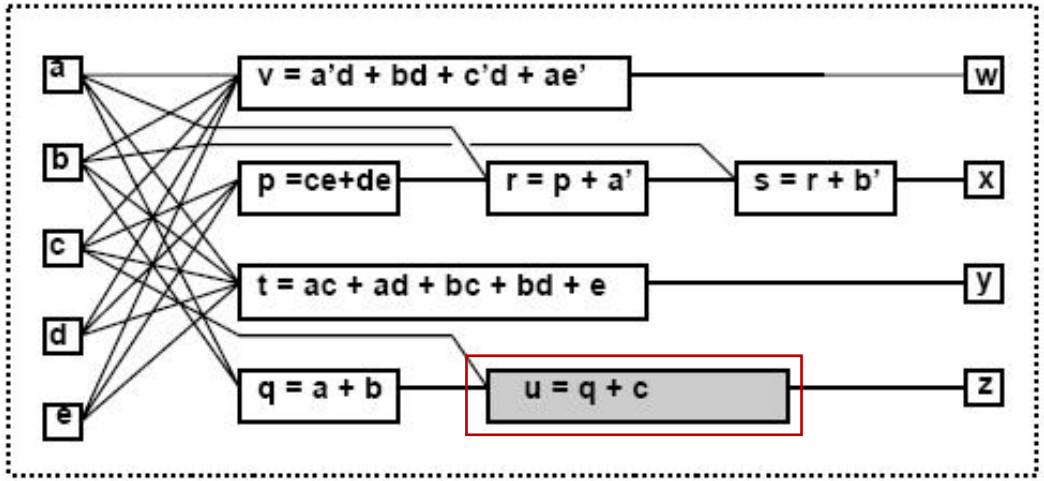
化简
Simplification

- 简化局部函数
  - 使用启发式最小化工具，如 Espresso
  - 修改目标节点的输入集合（fanin）

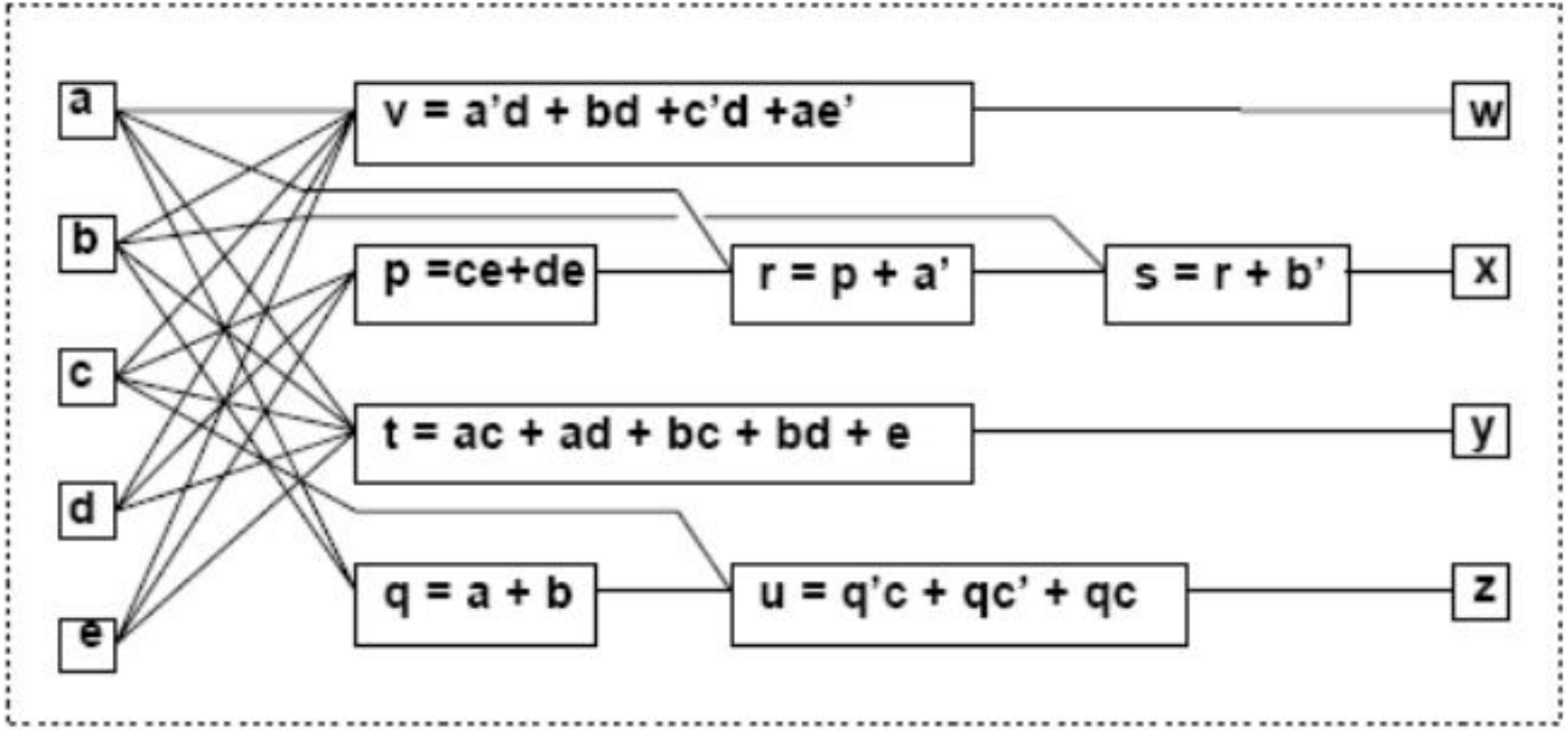
- Example:
  - u = q’c + qc’ + qc;
  - u = q + c;

例子


a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

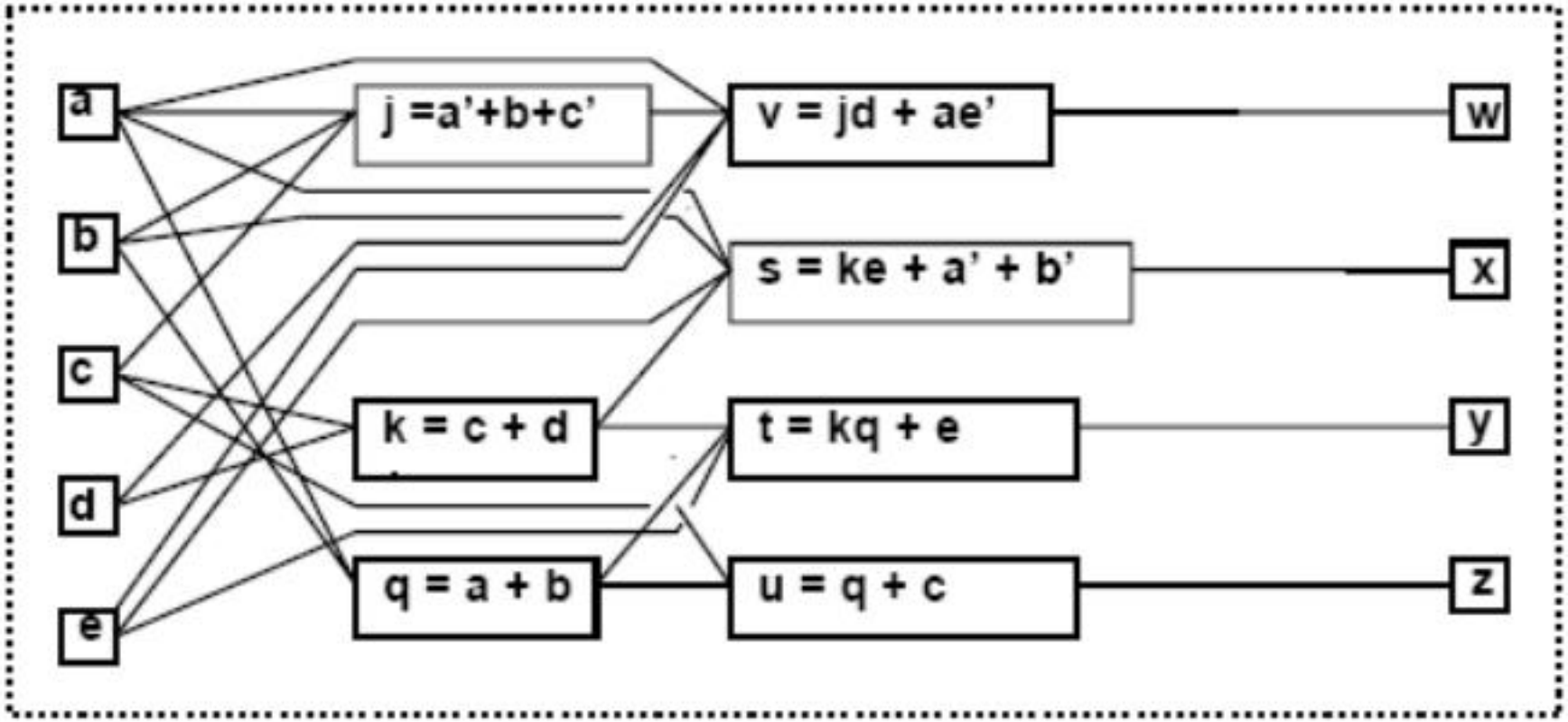

a
b
c
d
e
v = a'd + bd + c'd + ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q + c
w
x
y
z

Example – Sequence of transformations



布尔文字总数：33

Example – Sequence of transformations



布尔文字总数：20

估算指标

Optimization approaches

- 基于规则的方法
  (1) 建立规则数据库
  (2) 维护模式- 替换对
  (3) 按规则执行模式替换
- 算法式方法
  (1) 为每种转换类型定义算法
  (2) 算法充当网络上的操作符
  (3) 通过脚本依次调用这些算法
- 说明：现代综合工具主要采用算法式方法，但仍用规则处理特定问题。

代数法

布尔方法与代数方法

Boolean and algebraic methods

- 多级综合中的布尔方法
  - (1) 利用布尔函数特性
  - (2) 使用“不关心”条件
  - (3) 计算量大
- 代数方法
  - (1) 将逻辑函数抽象成多项式表示
  - (2) 实现更简单、速度更快，但优化能力较弱
  - (3) 应用广泛

布尔方法与代数方法

Boolean and algebraic methods

- 布尔代数
  - a) 取补运算
  - b) 对称分配律
  - c) 不关心集
- 代数建模思路
  - (1) 将布尔表达式视作多项式
  - (2) 使用乘积之和（Sum- of- Product, SOP）的形式
  - (3) 应用多项式代数

例子

- 布尔表示:
  - 给定: h = a + bcd + e; q = a + cd;
  - 可得: h = a + bq + e;
  - 因为: a + bq +e = a + b(a+cd) + e = a + bcd + e;
- 代数表示:
  - 给定: t = ka + kb + e; q = a + b;
  - 可得: t = kq + e;
  - 因为: kq = ka + kb;

代数模型

Algebraic model

- 给定两个代数表达式

- 当满足下列条件时，表达式 $f_{除式}$ 可“整除” $f_{被除式}$，且 $f_{商} = f_{被除式} / f_{除式}$：
  (1) $f_{被除式} = f_{除式} * f_{商} + f_{余式}$
  (2) $f_{除式} * f_{商} \neq 0$
  (3) $f_{除式}$ 与 $f_{商}$ 所包含的变量集合互不相交

- 注意：$f_{商}$ 与 $f_{除式}$ 在定义上是可互换的

例子

- 可以用代数除法的例子 :
  - $f_i = ac + ad + bc + bd + e$
  - $f_j = a + b$
  - 有 : $f_{商} = c + d$ and $f_{余式} = e$
    因为 $(a+b)\ (c+d) + e = f_i$
    且 $\{a,b\} \cap \{c,d\} = \emptyset$

除法的算法

An algorithm for division

- 符号说明
  - A：被除式的乘积集合 C^{A}_{j}，数量为 l
  - B：除式的乘积集合C^{B}_{i}，数量为 n
  - Q：商；R：余式

除法的算法

An algorithm for division

ALGEBRAIC_DIVISION(A,B)

{

for (i = 1 to n)

{

D = {C^A_j such that $C^A_j \supseteq C^B_i$ };

if (D == Ø) return(Ø,A);

D_i = D with variables in *sup*(C^B_i) dropped;

if (i = 1) Q = D_i ;

else Q = Q ∩ D_i ;

}

R = A – Q x B;

return(Q,R);

}

```
ALGEBRAIC_DIVISION(A, B)
  {
      对于 i 从 1 到 n  //即依次遍历除式B中的每一项
      {
          D = { C^A_j | C^A_j ⊇ C^B_i };
          如果 D 为 ø 则返回 (ø, A);
          Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;
          如果 i == 1
                  则 Q = Di;
          否则
                  Q = Q ∩ Di;
      }
      R = A – Q × B;
      返回 (Q, R);
  }
```

指C^{B}_{i}的输入，即所有出现过的变量组成的集合

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即： A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	1
C^B_i	a
D	

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即： A = {ac,ad,bc,bd,e} and B = {a,b}

```
ALGEBRAIC_DIVISION(A, B){
    对于 i 从 1 到 n  //即依次遍历除式B中的每一项 {
        D = { C^A_j | C^A_j ⊇ C^B_i };
        如果 D 为 Ø  则返回 (Ø, A);
        Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;
        如果 i == 1  则 Q = Di;
        否则  Q = Q ∩ Di; }
    R = A – Q × B;
    返回 (Q, R); }
```

变量	值
i	1
C^B_i	a
D	{ac, ad}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即： A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	1
C^B_i	a
D	{ac, ad}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即： A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	1
C^B_i	a
D	{ac, ad}
Di	{c,d}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即：A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	1
C^B_i	a
D	{ac, ad}
Di	{c,d}
Q	{c,d}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即： A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	2
C^B_i	b
D	{ac, ad}
Di	{c,d}
Q	{c,d}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即： A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	2
C^B_i	b
D	{bc,bd}
Di	{c,d}
Q	{c,d}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即： A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	2
C^B_i	b
D	{bc,bd}
Di	{c,d}
Q	{c,d}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即：A = {ac,ad,bc,bd,e} and B = {a,b}

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^{A}_{j} | $C^{A}_{j} \supseteq C^{B}_{i}$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^{B}_{i}) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	2
C^{B}_{i}	b
D	{bc,bd}
Di	{c,d}
Q	{c,d}

$f_{被除式}$ = ac+ad+bc+bd+e; $f_{除式}$ = a+b

即：A = {ac,ad,bc,bd,e} and B = {a,b}

R= ac+ad+bc+bd+e-(c+d)(a+b)
= ac+ad+bc+bd+e-ac-ad-bc-bd
= e

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	2
C^B_i	b
D	{bc,bd}
Di	{c,d}
Q	{c,d}
R	{e}

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

即：A = {ac,ad,bc,bd,e} and B = {a,b}

返回：（{c,d}，{e}）

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变量	值
i	2
C^B_i	b
D	{bc,bd}
Di	{c,d}
Q	{c,d}
R	{e}

例子

$f_{被除式} = ac+ad+bc+bd+e$; $f_{除式} = a+b$

- A = {ac,ad,bc,bd,e} and B = {a,b}
- i = 1:
  - C^B_1 = a, D = {ac,ad} and D_1 = {c,d}
  - Then Q = {c,d}
- i = 2 = n:
  - C^B_2 = b, D = {bc,bd} and D_2 = {c,d}
  - Then Q = {c,d} ∩ {c,d} = {c,d}
- Result:
  - Q = {c,d} and R = {e}
  - $f_{商}$ = c + d and $f_{余式}$ = e

快筛规则

- 给定代数表达式 fi 和 fj，如果出现以下任一情况，则 fi / fj 的结果为空：

  1. fj 包含 fi 中未出现的变量

  例子：

  fi = ab+bc

  fj = a+d

快筛规则

- 给定代数表达式 fi 和 fj，如果出现以下任一情况，则 fi / fj 的结果为空：

  2. fj 含有某个乘积，其变量集不被 fi 中任何乘积的变量集包含

  例子：

  fi = ab+bc

  fj = ac+b

快筛规则

- 给定代数表达式 fi 和 fj，如果出现以下任一情况，则 fi / fj 的结果为空：

  3. fj 的项数多于 fi

  例子：

  fi = ab+bc

  fj = a+b+c

快筛规则

- 给定代数表达式 fi 和 fj，如果出现以下任一情况，则 fi / fj 的结果为空：

  4. fj 中某个变量的数量高于 fi

  例子：

  fi = ab+bc

  fj = a+ab

快筛规则

- 给定代数表达式 fi 和 fj，如果出现以下任一情况，则 fi / fj 的结果为空:

  1. fj 包含 fi 中未出现的变量
  2. fj 含有某个乘积，其变量集不被 fi 中任何乘积的变量集包含
  3. fj 的项数多于 fi
  4. fj 中某个变量的数量高于 fi

作业

- 假设被除式为f=abc+acd+abd+e，除式为g=ab+ad
- 请用代数除法计算f/g的商和余式应该是多少

ALGEBRAIC_DIVISION(A, B){

对于 i 从 1 到 n //即依次遍历除式B中的每一项 {

D = { C^A_j | $C^A_j \supseteq C^B_i$ };

如果 D 为 Ø 则返回 (Ø, A);

Di = 用 D 中每一项删去 sup(C^B_i) 中的变量所得的结果;

如果 i == 1 则 Q = Di;

否则 Q = Q ∩ Di; }

R = A – Q × B;

返回 (Q, R); }

变换方式总结

1.消除

- 执行变量替换操作

2.拆分

- 将一个函数拆分为更小的函数

3.提取

- 在两个（或多个）表达式中查找公共子表达式

4.替代

- 通过引入一个原本不属于原输入集的已有布尔函数来简化局部函数

5.化简

- 两级逻辑优化

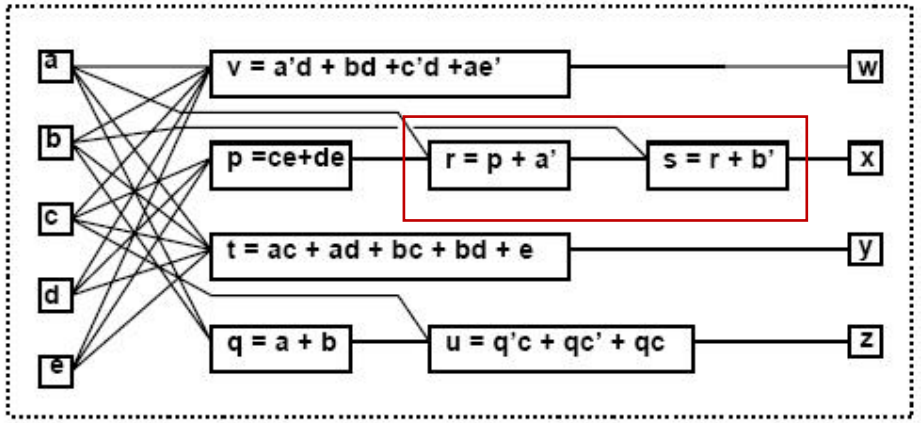
消除

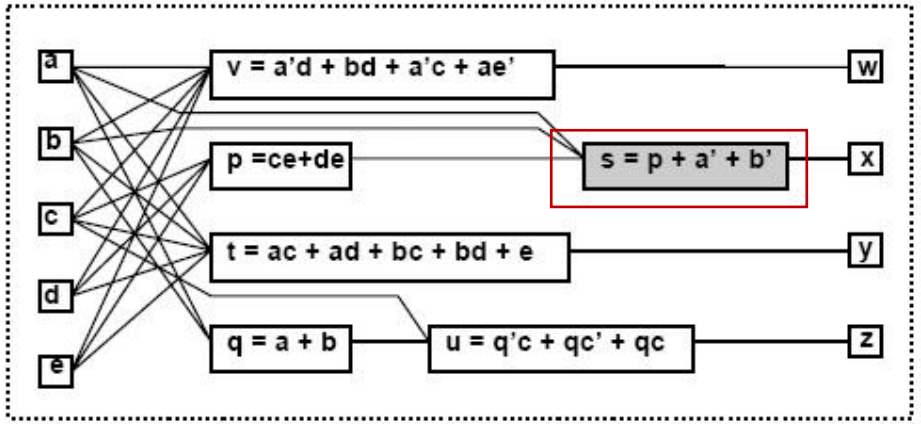
Elimination

- 从网络中消除一个函数
  - 类似于高斯消元
- 执行变量替换操作

- Example:
  - s = r + b’; r = p + a’;
  - s = p + a’ + b’;

例子


a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z


a
b
c
d
e
v = a'd + bd + a'c + ae'
p =ce+de
s = p + a' + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

变换方式总结

1.消除

- 执行变量替换操作

2.拆分

- 将一个函数拆分为更小的函数

3.提取

- 在两个（或多个）表达式中查找公共子表达式

4.替代

- 通过引入一个原本不属于原输入集的已有布尔函数来简化局部函数

5.化简

- 两级逻辑优化

利用代数法完成替代操作

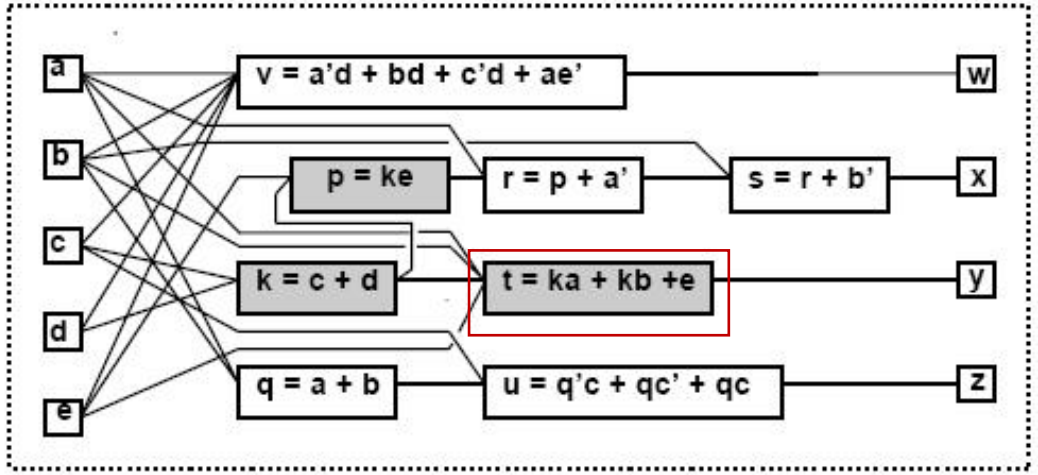
替代

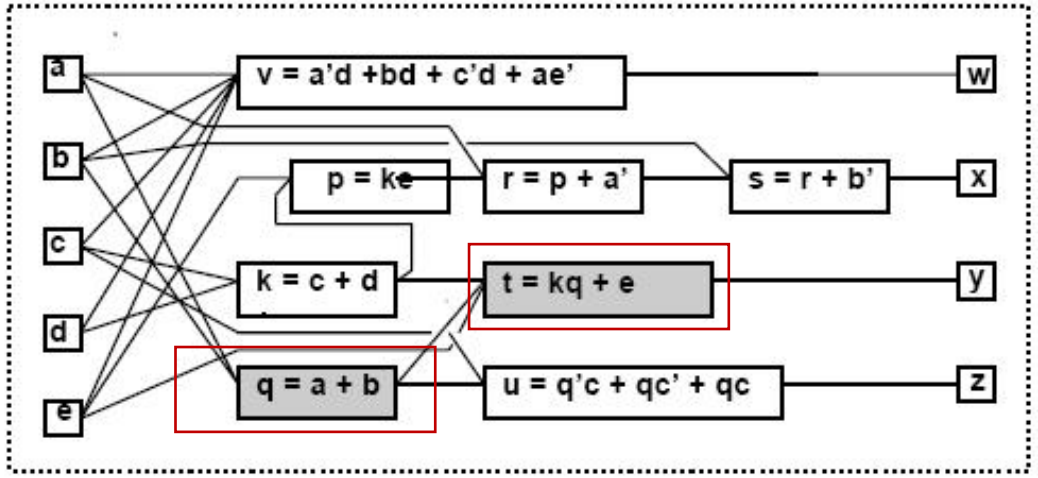
Substitution

- 通过引入一个原本不属于原输入集的附加输入来简化局部函数

- Example:
  - t = ka + kb + e;
  - t = kq + e;
  - Because q = a + b is already part of the network

例子


a
b
c
d
e
v = a'd + bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = ka + kb +e
q = a + b
u = q'c + qc' + qc
w
x
y
z


a
b
c
d
e
v = a'd +bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = kq + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

代数替代

Algebraic substitution

- 选取一对表达式
- 以任意顺序对它们执行除法
- 若所得商（quotient）非空：
  - 评估面积收益和时间收益
  - 将 $f_{被除式}$ 替换为 $j * f_{商} + f_{余式}$，其中 j 是与 $f_{除式}$ 对应的变量
- 依据前述定理使用快筛规则，减少不必要的计算

```
SUBSTITUTE(Gn(V,E)){
  for (i = 1,2,…,|V|){
      for (j = 1,2,…,|V|;j ≠ i){
          A = set of cubes of f_i;
          B = set of cubes of f_j;
          if (A,B pass the filter test){
              (Q,R) = ALGEBRAIC_DIVISION(A,B);
              if (Q ≠ Ø){
                  f_商 = sum of cubes of Q;
                  f_余式 = sum of cubes of R;
                  if (substitution is favorable)
                      f_i = j * f_商 + f_余式;
              }
          }
      }
  }
}
```

```
SUBSTITUTE(Gₙ (V, E))
  {
      对于 i = 1, 2, ···, |V|
      {
          对于 j = 1, 2, ···, |V|（且 j ≠ i）
          {
              A = fᵢ 的乘积集合；
              B = fⱼ  的乘积集合；
              如果 (A, B) 通过快筛规则仍然保留
              {
                  (Q, R) = ALGEBRAIC_DIVISION(A, B);
                  如果 Q ≠ Ø
                  {
                      f商 = Q 中所有乘积之和;
                      f余式 = R 中所有乘积之和;
                      如果替换是有利的
                          fi = j * f商 + f余式;
                  }
              }
          }
      }
  }
```

V = {ka + kb + e, a+b};

```
SUBSTITUTE(Gn  (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj  的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式;  }}}}}
```

变量	值
i	1
j	2
A	{ka,kb,e}
B	{a,b}
(Q,R)	

V = {ka + kb + e, a+b};

不能出现以下情况：

1. fj 包含 fi 中未出现的变量
2. fj 含有某个乘积，其变量集不被 fi 中任何乘积的变量集包含
3. fj 的项数多于 fi
4. fj 中某个变量的数量高于 fi

```
SUBSTITUTE(Gn (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj 的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式；  }}}}}
```

变量	值
i	1
j	2
A	{ka,kb,e}
B	{a,b}
(Q,R)	

V = {ka + kb + e, a+b};

```
SUBSTITUTE(Gn  (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj  的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式;  }}}}}
```

变量	值
i	1
j	2
A	{ka,kb,e}
B	{a,b}
(Q,R)	(k,e)

V = {ka + kb + e, a+b};

```
SUBSTITUTE(Gn  (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj  的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式;  }}}}}
```

变量	值
i	1
j	2
A	{ka,kb,e}
B	{a,b}
(Q,R)	(k,e)

V = {ka + kb + e, a+b};

```
SUBSTITUTE(Gn (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj 的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式；  }}}}}
```

变量	值
i	1
j	2
A	{ka,kb,e}
B	{a,b}
(Q,R)	(k,e)
f商	k
f余式	e

V = {2*k+e, a+b};

原式：f=ka+kb+e　　布尔文字数：5

新式：f=2*k+e　　布尔文字数：3

```
SUBSTITUTE(Gn (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj 的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式;  }}}}}
```

变量	值
i	1
j	2
A	{ka,kb,e}
B	{a,b}
(Q,R)	(k,e)
f商	k
f余式	e

V = {2*k+e, a+b};

原式：f=ka+kb+e　　布尔文字数：5

新式：f=2*k+e　　布尔文字数：3

```
SUBSTITUTE(Gn (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj 的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式；  }}}}}
```

变量	值
i	2
j	1
A	{a,b}
B	{2k,e}
(Q,R)	(k,e)
f商	k
f余式	e

V = {2*k+e, a+b};

给定代数表达式 fi 和 fj，如果出现以下任一情况，则 fi / fj 的结果为空：

1. fj 包含 fi 中未出现的变量

```
SUBSTITUTE(Gn (V, E)){
    对于 i = 1, 2, ···, |V|{
        对于 j = 1, 2, ···, |V|（且 j ≠ i）{
            A = fi 的乘积集合；
            B = fj 的乘积集合；
            如果 (A, B) 通过快筛规则仍然保留{
                (Q, R) = ALGEBRAIC_DIVISION(A, B);
                如果 Q ≠ Ø{
                    f商 = Q 中所有乘积之和;
                    f余式 = R 中所有乘积之和;
                    如果替换是有利的
                        fi = j * f商 + f余式；}}}}}
```

变量	值
i	2
j	1
A	{a,b}
B	{2k,e}
(Q,R)	(k,e)
f商	k
f余式	e

代数替代

Algebraic substitution

- 选取一对表达式
- 以任意顺序对它们执行除法
- 若所得商（quotient）非空：
  - 评估面积收益和时间收益
  - 将 $f_{被除式}$ 替换为 $j * f_{商} + f_{余式}$，其中 j 是与 $f_{除式}$ 对应的变量
- 依据前述定理使用快筛规则，减少不必要的计算

变换方式总结

1.消除

- 执行变量替换操作

2.拆分

- 将一个函数拆分为更小的函数

3.提取

- 在两个（或多个）表达式中查找公共子表达式

4.替代

- 通过引入一个原本不属于原输入集的已有布尔函数来简化局部函数

5.化简

- 两级逻辑优化

定义

Definitions

- Cube-free（非乘积）表达式
  - 指无法再被任何项因式分解的表达式（即无法写成两个式子相乘的表达式）

- 例子:
  - a + bc 是Cube-free表达式
  - abc 以及 ab + ac 都不是Cube-free表达式

定义

Definitions

- 表达式的kernel（核）
  - 将表达式除以某个项（该项会被称为co-kernel，共核）后得到的Cube-free商，称为核
  - “共核”和“核”是一一对应的
- 表达式 f 的核集合记作 K(f)

核集的计算

Kernel set computation

- 朴素方法
  - 用变量集中所有子集（其幂集）的元素去除原函数。
  - 剔除非cube- free的商。

例子

f = ace + bce + de + g

核搜索：

- 用 a 除 f，得 ce，非Cube-free表达式
- 用 b 除 f，得 ce，非Cube-free表达式
- 用 c 除 f，得 ae + be，非Cube-free表达式
- 用 d 除 f，得 e，非Cube-free表达式
- 用 e 除 f，得 ac + bc + d，为Cube-free表达式 → 核
- 用 g 除 f，得 1，非Cube-free表达式
- 用 ab无法除f, 根据快筛规则
- …
- 用 ce 除 f，得 a + b，为Cube-free表达式 → 核
- …
- 用 1 除 f，得 ace + bce + de + g，为Cube-free表达式 → 核

- K(f) = { a + b，ac + bc + d，ace + bce + de + g }
- CoK(f) = { ce，e，1 }

核集的计算

Kernel set computation

- 朴素方法
  - 用变量集中所有子集（其幂集）的元素去除原函数。
  - 剔除非cube- free的商。

  - 改进方法
  - 用递归算法减少计算
  - 利用乘法的交换律来减少重复计算。