两级逻辑优化

Two-level Logic Optimization

2025-5

主讲人：孙静翎

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

启发式最小化的基本原则

Heuristic minimization -- principles

- 从初始覆盖出发
  - 由设计者提供，或从硬件语言模型中提取
- 对当前覆盖进行修改
  - 使其成为质覆盖、无冗余覆盖
  - 重复迭代，直到获得较小的无冗余覆盖
- 通常覆盖的规模会减少
  - 对小规模覆盖的操作速度快

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

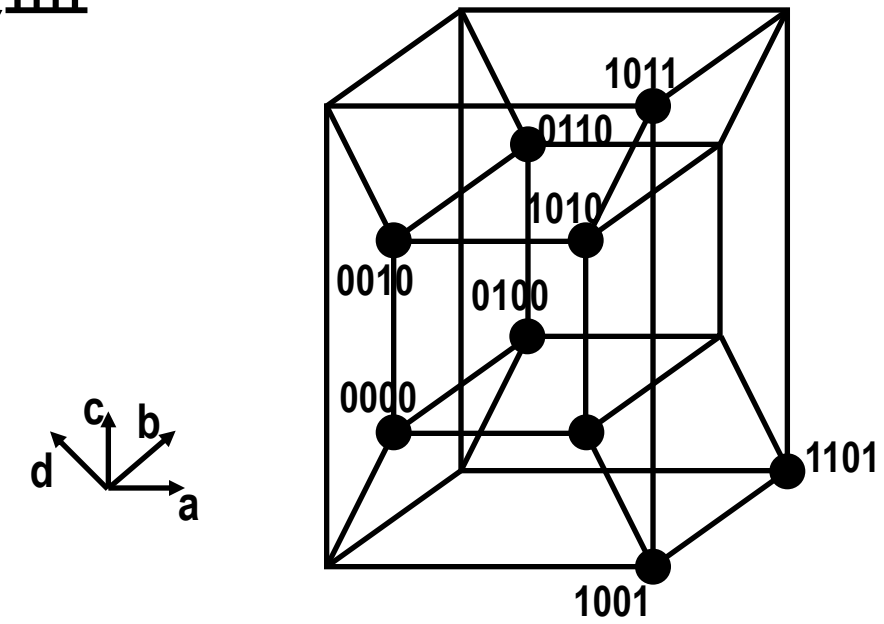
Heuristic minimization - operators

- 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

例子

F=a’b’c’d’+a’b’cd’+a’bc’d’+a’bcd’+ab’c’d’+ab’cd’+ab’c’d+ab’cd+abc’d

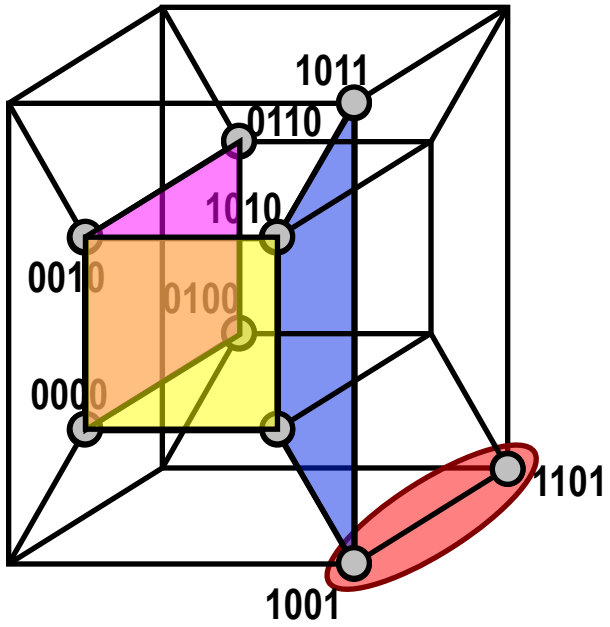
- 初始覆盖

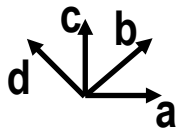


MINI的操作步骤-例子

- 1.扩展:

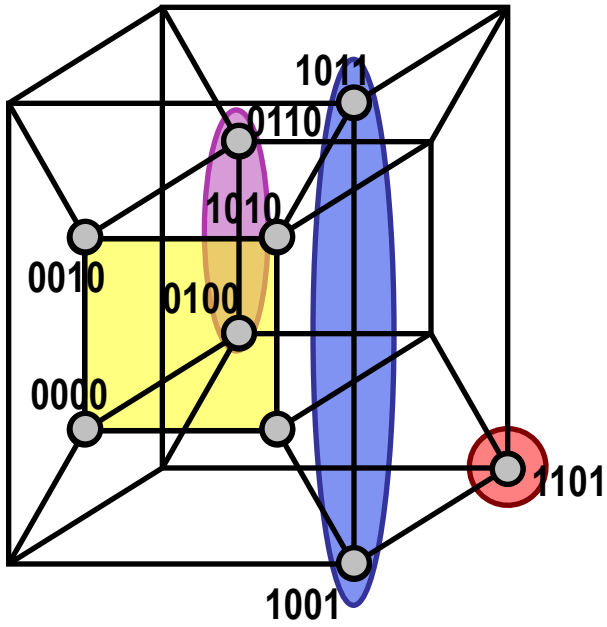
- 将 0000 扩展为 α = 0**0.
  - 删除被0**0覆盖的 0100, 0010, 0110.
- 将1000扩展为β = *0*0.
  - 删除被*0*0覆盖的1010.
- 将1001扩展为δ = 10**.
  - 删除被10**覆盖的1011.
- 将1101扩展为ε = 1*01.
- 获得的覆盖是: {α, β, δ, ε}.



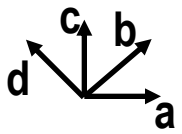


MINI的操作步骤-例子

- 1. 扩展:
  - 获得的覆盖是: {α,β,δ,ε}.
  - 质覆盖, 单个蕴含项下的非冗余覆盖.
- 2. 缩减:

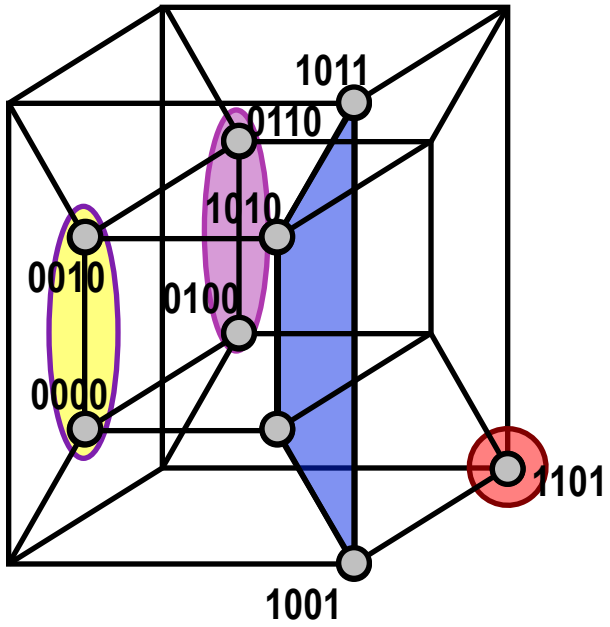


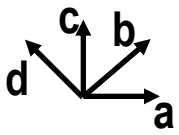
- 缩减α = 0**0为α′ = 01*0
- 缩减 δ = 10** 为 δ’ = 10*1.
- 缩减ε = 1*01 为 ε’ = 1101.
- 获得的覆盖是: {α′ β δ′ ε’ }.



MINI的操作步骤-例子

- 1. 扩展:
  - 获得的覆盖是: {α,β,δ,ε}.
  - 质覆盖, 单个蕴含项下的非冗余覆盖.
- 2. 缩减:
  - 缩减α = 0**0为α′ = 01*0
  - 缩减 δ = 10** 为 δ′ = 10*1.
  - 缩减ε = 1*01 为 ε′ = 1101.
  - 获得的覆盖是: {α′ β,δ′,ε′}.
- 3. 重构:

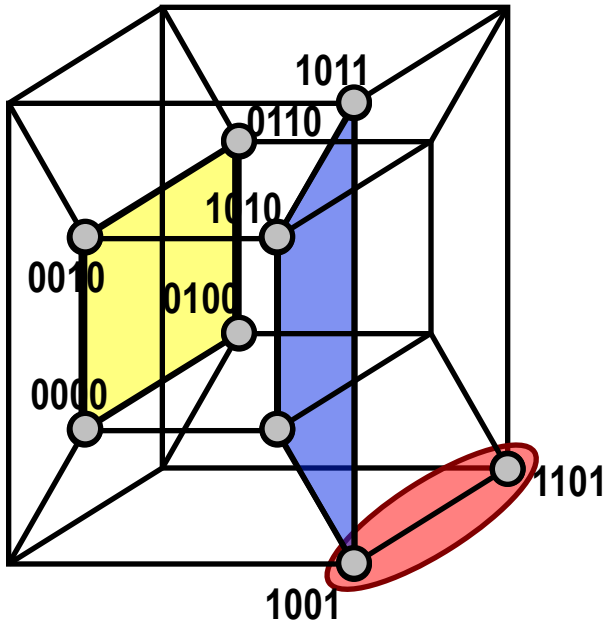


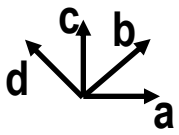


- 重构 {β, δ′} 为: {β′, δ}.
  - 其中 β′ = 00*0.
- 获得的覆盖是: {α′ β′, δ, ε’ ’ }.

MINI的操作步骤-例子

- 1. 扩展:
  - 获得的覆盖是: {α,β,δ,ε}.
  - 质覆盖, 单个蕴含项下的非冗余覆盖.
- 2. 缩减:
  - 缩减α = 0**0为α′ = 01*0
  - 缩减 δ = 10** 为 δ′ = 10*1.
  - 缩减ε = 1*01 为 ε′ = 1101.
  - 获得的覆盖是: {α′ β,δ′,ε′}.
- 3. 重构:
  - 重构 {β, δ'} 为: {β', δ}，其中 β' = 00*0.
  - 获得的覆盖是: {α' β',δ,ε'}.
- 4. 二次扩展:

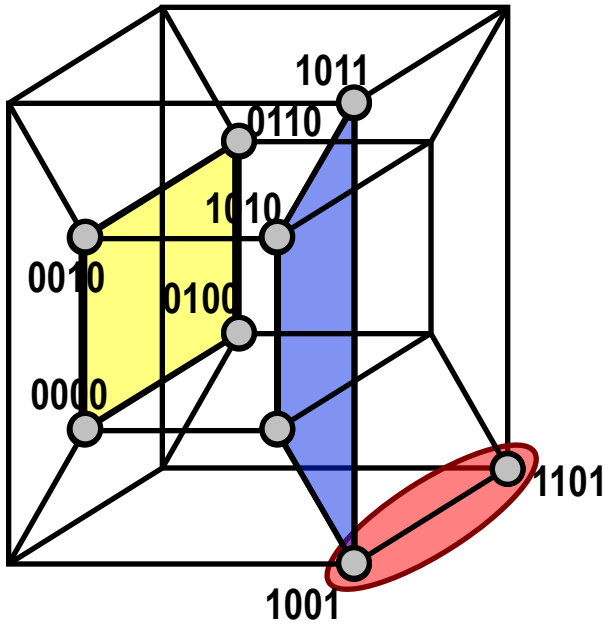




- 扩展 β = 00*0 为 β = 0**0，删除被0**0覆盖的01*0
- 扩展 ε’ = 1101 为 ε = 1*01.

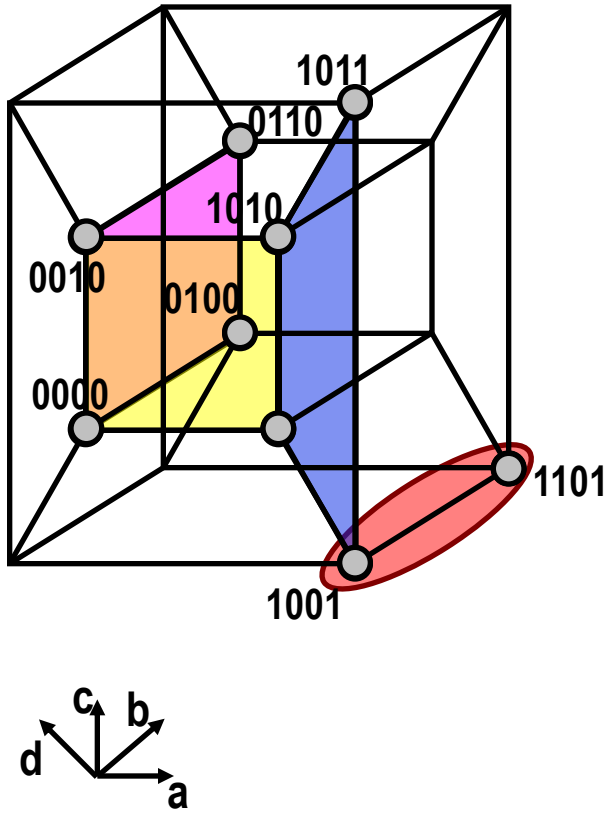
MINI的操作步骤-例子

- 1. 扩展:
  - 获得的覆盖是: {α,β,δ,ε}.
  - 质覆盖, 单个蕴含项下的非冗余覆盖.
- 2. 缩减:
  - 缩减α = 0**0为α′ = 01*0
  - 缩减 δ = 10** 为 δ′ = 10*1.
  - 缩减ε = 1*01 为 ε′ = 1101.
  - 获得的覆盖是: {α′ β,δ′,ε′}.
- 3. 重构:
  - 重构 {β, δ'} 为: {β', δ}，其中 β' = 00*0.
  - 获得的覆盖是: {α' β',δ,ε'}.
- 4. 二次扩展:
  - 扩展 β = 00*0 为 β = 0**0，删除被0**0覆盖的01*0
  - 扩展 ε' = 1101 为 ε = 1*01.



ESPRESSO的操作步骤-例子

- 扩展:
  - 覆盖: {α, β, δ, ε}.
  - 质覆盖, 单个蕴含项下的非冗余覆盖.
- 去冗余:
  - 覆盖: {α, δ, ε}.
  - 质覆盖, 非冗余覆盖.



扩展 – 简单实现

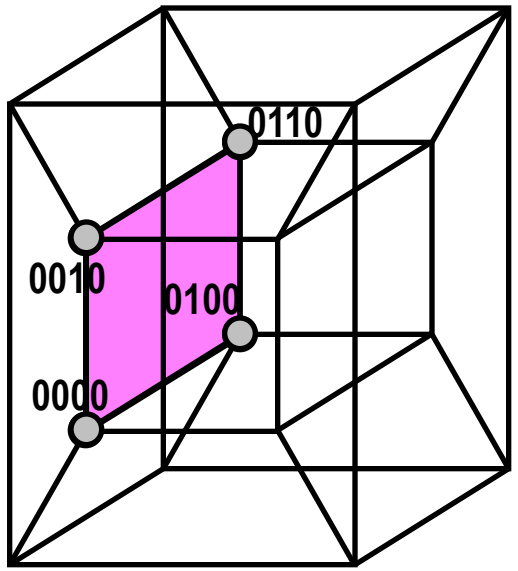
Expand - Naïve implementation

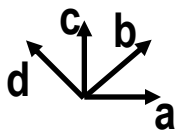
- 对于每个蕴含项
  - 对其中每个不为“无关项”（don’t care）的布尔文字
  - 如果可能，将其更改为“无关项”
  - 移除所有被扩展后的蕴含项覆盖的蕴含项

扩展的例子

F=a’b’c’d’+a’b’cd’+a’b’cd’+a’bcd’

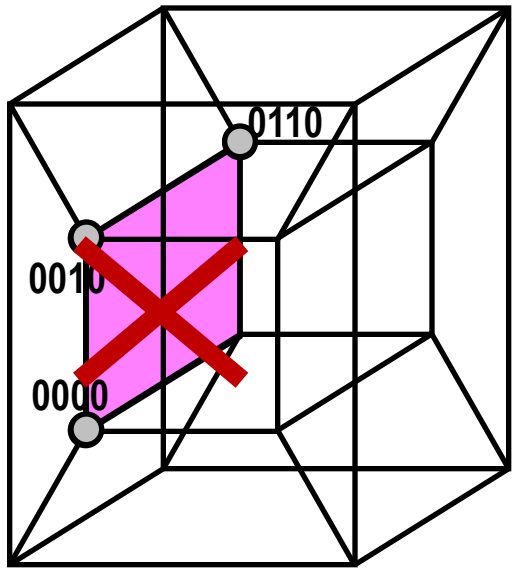
- Expand 0000 to α = 0**0.
  - Drop 0100, 0010, 0110 from the cover.

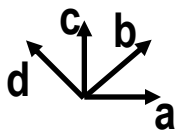




扩展的例子

F=a’b’c’d’+a’b’cd’+a’bcd’





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

作业

请用矩阵表示法确认以下函数的a’bcd’能否扩展为a’bc：

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

Espresso、 MINI

- 检查扩展后蕴含项与 F’ 的交集是否为空
- Presto
- 检查扩展后蕴含项是否包含于 F

例子

F = a’bcd+a’bcd’+a’b’cd

1. 求F’

对负单调变量a取余因子

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

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

F_a:

10 01 01 01

10 01 01 10

10 10 01 01

01 11 11 11

---

00 01 01 01

00 01 01 10

00 10 01 01

F_a=0 -> F'_a=1

$F_{a'}$:

10 01 01 01

10 01 01 10

10 10 01 01

10 11 11 11

---

10 01 01 01

10 01 01 10

10 10 01 01

01 00 00 00

---

11 01 01 01

11 01 01 10

11 10 01 01

例子

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

1. 求F'

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

对正单调变量c取余因子

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

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

$F_{a'c}$:

11 01 01 01

11 01 01 10

11 10 01 01

11 11 01 11

---

11 01 01 01

11 01 01 10

11 10 01 01

00 00 10 00

---

11 01 11 01

11 01 11 10

11 10 11 01

$F_{a'c'}$:

11 01 01 01

11 01 01 10

11 10 01 01

11 11 10 11

---

11 01 00 01

11 01 00 10

11 10 00 01

$F_{a'c'}=0 \rightarrow F'_{a'c'}=1$

例子

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

1. 求F'

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

对变量b取余因子

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

$=a+0+b'\ F'_{a'cb'} +c'$

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

$F_{a'cb}$: 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

$F_{a'cb}=1 \rightarrow F'_{a'cb}=0$

$F_{a'cb'}$: 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

$F_{a'cb'}$: 11 11 11 01

$F'_{a'cb'}$:11 11 11 10

例子

F = a’bcd+a’bcd’+a’b’cd

1. 求F’

F’= a+b’d’+c’

2. 求F’和a’bc交集是否为空

交集为空，所以a’bcd’能扩展为a’bc

01 11 11 11

11 10 11 10

11 11 10 11

10 01 01 11

---

00 01 01 11

11 00 01 11

10 01 00 11

作业

请用矩阵表示法确认以下函数的a’bcd’能否扩展为a’bc：

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

Espresso、 MINI

- 检查扩展后蕴含项与 F’ 的交集是否为空
- Presto
- 检查扩展后蕴含项是否包含于 F

例子 - Presto

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

检查a'bcd'能否扩展为a'bc：

1. 计算F是否包含a'bc

$F_{a'bc}$为重言式，F包含a'bc
-> a'bcd'可以扩展为a'bc

$F_{a'bc}$: 10 01 01 01
10 01 01 10
10 10 01 01
10 01 01 11

---

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

---

11 11 11 01
11 11 11 10

重言式

正式开始本周的内容

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

启发式排序

Ordering heuristics

扩展不太可能被其他“立方体”（cubes）覆盖的立方体

- 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

例子

- f = a’b’c’ + ab’c’ + a’bc’ + a’b’c

- 排序:

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

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

    9（a’b’c’）,7（ab’c’）,7（a’bc’）,7（a’b’c)

  - 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ab’c’或a’bc’或a’b’c

启发式排序

Ordering heuristics

扩展不太可能被其他“立方体”（cubes）覆盖的立方体

- 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）

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

例子

$$f = a'b'c' + ab'c' + a'bc' + a'b'c$$

例子

$f = a'b'c' + ab'c' + a'bc' + a'b'c$

1. 求F'

对变量a取余因子

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

$= a(b+c) + a'F'_{a'}$

F_a: 10 10 10
01 10 10
10 01 10
10 10 01
01 11 11
---
00 10 10
01 10 10
00 01 10
00 10 01
10 00 00

F_a: 11 10 10

$F_{a'}$: 10 10 10
01 10 10
10 01 10
10 10 01
10 11 11
---
10 10 10
00 10 10
10 01 10
10 10 01
01 00 00
---
11 10 10
11 01 10
11 10 01

F'_a: 11 01 11
11 11 01

例子

$f = a'b'c' + ab'c' + a'bc' + a'b'c$

1. 求F'

对变量a取余因子

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

$= a(b+c) + a'F'_{a'}$

$= a(b+c) + a'(b\,F'_{a'b} + b'\,F'_{a'b'})$

$= ab + ac + a'(bc+0)$

$= ab + ac + a'bc$

$F_{a'b}$: 11 10 10
11 01 10
11 10 01
11 01 11
---
11 00 10
11 01 10
11 00 01
00 10 00

$F_{a'b}$: 11 11 10

$F'_{a'b}$: 11 11 01

$F_{a'b'}$: 11 10 10
11 01 10
11 10 01
11 10 11
---
11 10 10
11 00 10
11 10 01
00 01 00
---
11 11 10
11 11 01

$F_{a'b'}=1 \rightarrow F'_{a'b'}=0$

例子

- f = a’b’c’ + ab’c’ + a’bc’ + a’b’c

- 排序:

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

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

    9（a’b’c’）,7（ab’c’）,7（a’bc’）,7（a’b’c)

  - 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ab’c’或a’bc’或a’b’c

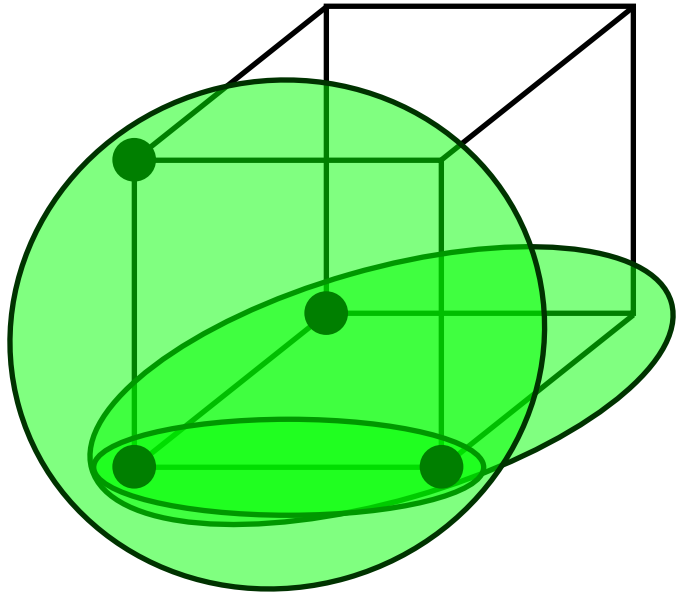
例子

- F:

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

F’:

01	01	11
01	11	01
10	01	01



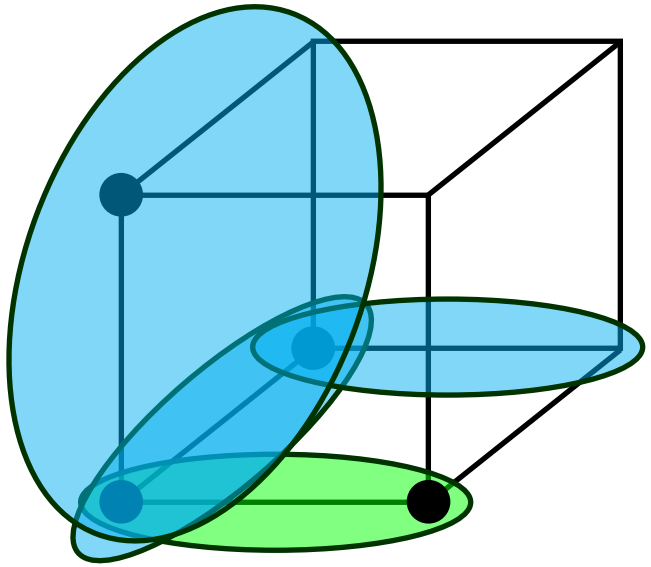
- 扩展ab’c’（01 10 10）：
  - 扩展为b’c’（11 10 10） 有效.
  - 扩展为c’（11 11 10） 无效.
  - 扩展为b’（11 10 11） 无效.
- 最终扩展ab’c’为b’c’，移除所有被扩展后的蕴含项覆盖的蕴含项：
  - ab’c’、a’b’c’
- 更新后的覆盖

11	10	10
10	01	10
10	10	01

例子

- F: 11 10 10
  10 01 10
  10 10 01

  F': 01 01 11
  01 11 01
  10 01 01



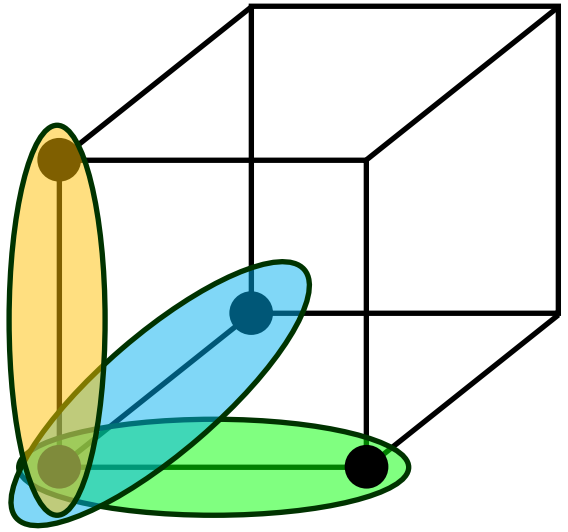
- 扩展a'bc'（10 01 10）：
  - 扩展为bc'（11 01 10）无效.
  - 扩展为a'c'（10 11 10） 有效.
  - 扩展为a'（10 11 11） 无效.
- 最终扩展a'bc' 为a'c'，移除所有被扩展后的蕴含项覆盖的蕴含项：
  - a'bc'
- 更新后的覆盖

11 10 10
10 11 10
10 10 01

例子

- F:
  11 10 10
  10 11 10
  10 10 01

  F’:
  01 01 11
  01 11 01
  10 01 01



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

  11 10 10
  10 11 10
  10 10 11

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

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

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

  - 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

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

例子

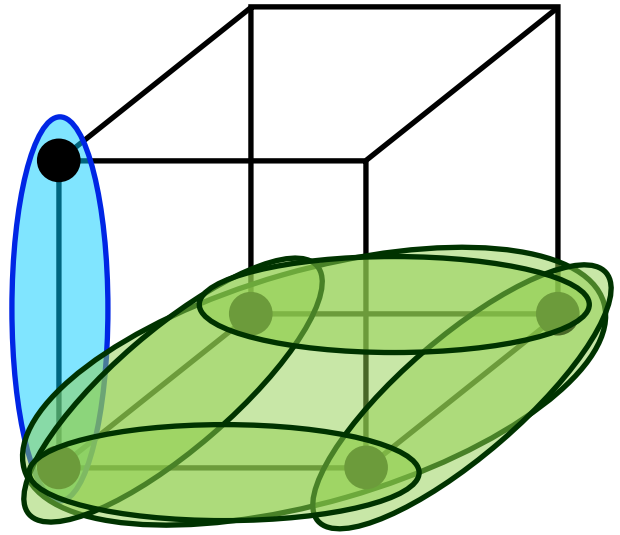
- F=c’+a’b’ **11 11 10**
  10 10 11
- 选择第一个蕴含项：
  - 尝试：

11 11 10 - > 10 11 10

11 11 10 - > 01 11 10

11 11 10 - > 11 10 10

11 11 10 - > 11 01 10

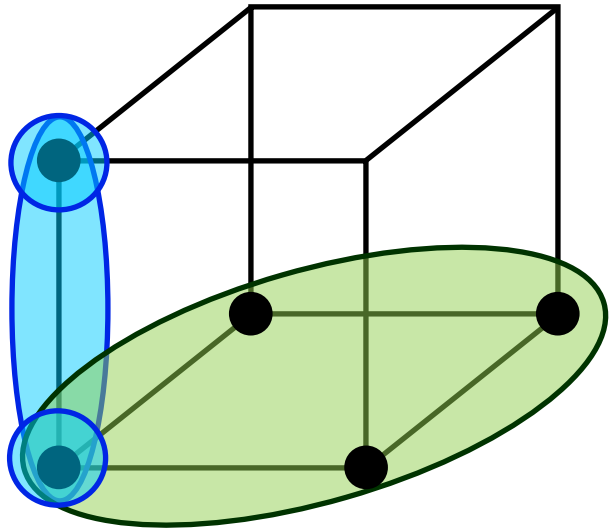


例子

- 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：

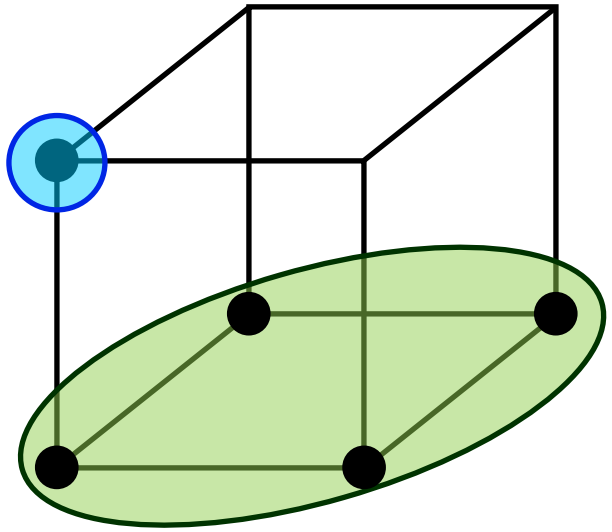
  10 10 11- > 10 10 10

  10 10 11- > 10 10 01



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 – 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

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：

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

例子

- F=c'+a'b'

11	11	10
10	10	11

- 选择第一个蕴含项(c')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'
  - Q'=a+b
  - Q'α: a+b
  - supercube(Q'α)=1
  - á = α ∩ supercube(Q'α)=c' (即无法化简)

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

例子

- F=c'+a'b'

11 11 10
10 10 11

- 选择第二个蕴含项(a'b')作为α:

ά = α ∩ supercube(Q'对α取余因子)

  - Q=除α以外的剩余的蕴含项相加 = c'
  - Q'=c
  - Q'α: c
  - supercube(Q'α)=c
  - ά = α ∩ supercube(Q'α)=a'b'c (最优化简)

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

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'b'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

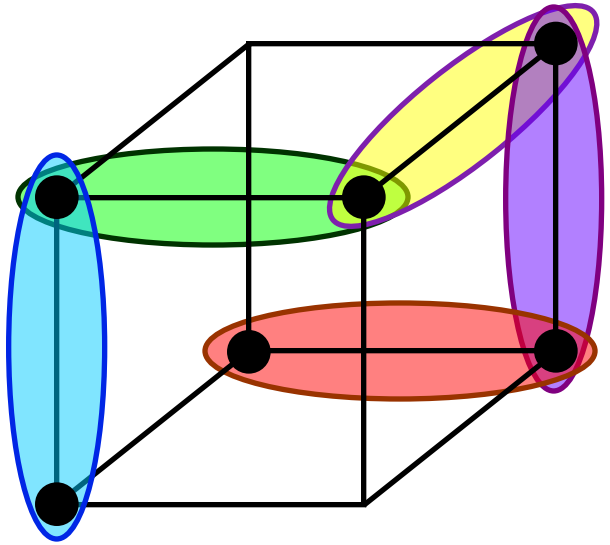
---

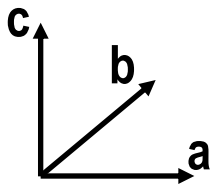
11 11 11 11
11 11 11 10

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

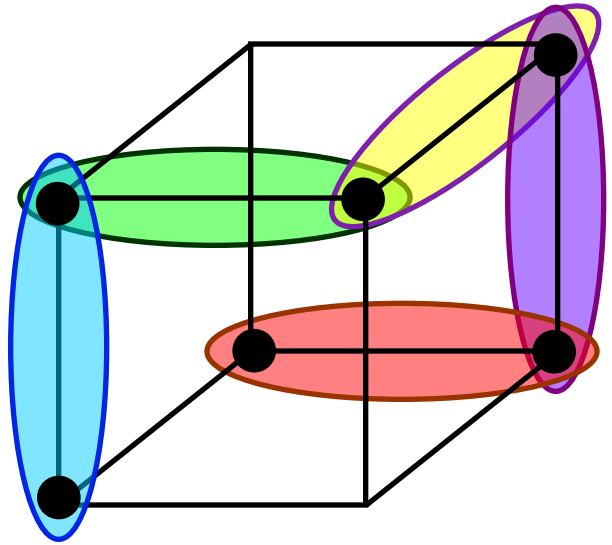
Irredundant

- 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。

例子

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

- $E^r = \{\alpha, \varepsilon\}$
- $R^t = \emptyset$
- $R^p = \{\beta, \gamma, \delta\}$



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)

例子

F=a’b’+b’c+ac

判断a’b’是否是相对必要项

Q=b’c+ac

$Q_{a'b'}$不为重言式=>a’b’是相对必要项

Er = {a’b’}

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

例子

F=a’b’+b’c+ac

判断b’c是否是相对必要项

Q=a’b’+ac

$Q_{b'c}$为重言式=>b’c不是相对必要项

Er = {a’b’}

10 10 11
01 11 01
11 10 01

---

10 10 01
01 10 01
00 01 10

---

10 11 11
01 11 11

例子

F=a’b’+b’c+ac

判断ac是否是相对必要项

Q=a’b’+b’c

Q_{ac}不为重言式=>ac是相对必要项

Er = {a’b’, ac}

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

例子

F=a’b’+b’c+ac

Er = {a’b’, ac}

判断b’c是否是完全冗余项

Er包含b’c=>b’c是完全冗余项

Er = {a’b’, ac}

Rt = {b’c}

Rp = ∅

10 10 11

01 11 01

11 10 01

---

10 10 01

01 10 01

00 01 10

---

10 11 11

01 11 11

作业（先不用交）

请用矩阵表示法为以下布尔函数划分Er、Rt和Rp：

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

- 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)

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

去冗余

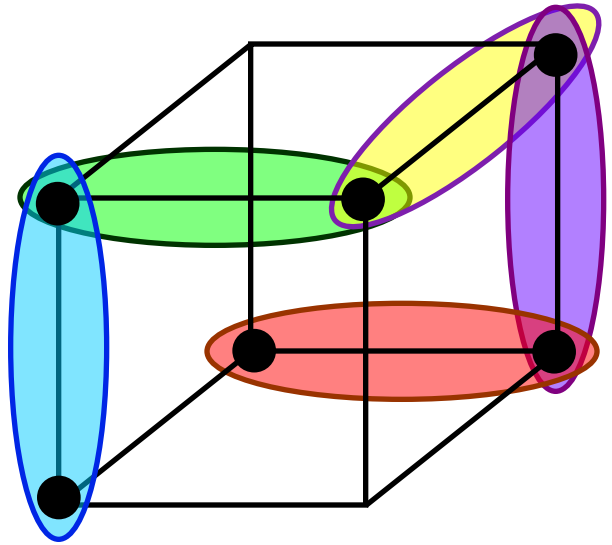
Irredundant

- 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

例子

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

- $E^r = \{\alpha, \varepsilon\}$
- $R^t = \emptyset$
- $R^p = \{\beta, \gamma, \delta\}$



- $E^r = \{\alpha, \varepsilon\}$
- $R^t = \emptyset$
- $R^p = \{\beta, \gamma, \delta\}$

例子

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

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

10 11 11
01 11 11

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

- 判断是否可以去掉β:
  - ➢H=Rp+Er-{β}=a'b'+ac+ab+bc'
  - ➢H_β是重言式 = 》可以去掉β

- $E^r = \{\alpha, \varepsilon\}$
- $R^t = \emptyset$
- $R^p = \{\gamma, \delta\}$

例子

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

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

- 判断是否可以去掉γ：
  - ➢H=Rp+Er-{α}=a'b'+ab+bc'
  - ➢H_γ不是重言式 = 》不可以去掉γ

- $E^r = \{\alpha, \varepsilon\}$
- $R^t = \emptyset$
- $R^p = \{\gamma, \delta\}$

例子

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

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

- 判断是否可以去掉δ:

➢H=Rp+Er-{α}=a'b'+ac+bc'

➢H_δ是重言式 = 》可以去掉δ

最终得到的非冗余覆盖：{α, ε,γ}

ESPRESSO

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算法

ESPRESSO algorithm

```
espresso(F,D) {
        R = complement(F U D);
        F = expand(F,R);
        F = irredundant(F,D);
        E = essentials(F,D);
        F = F – E;  D = D U E;
        repeat {
                φ2 = cost(F);
                repeat {
                        φ1 = |F |;
                        F = reduce(F,D);
                        F = expand(F,R);
                        F = irredundant(F,D);
                } until (|F | ≥ φ1);
                F = last_gasp(F,D,R);
        } until (cost( F ) ≥ φ2);
        F = F U E;  D = D – E;
        F = make_sparse(F,D,R);
}
```

```
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);
}
```

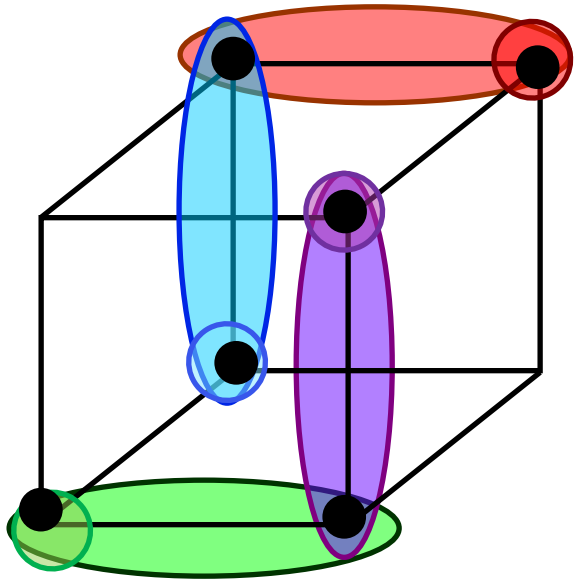
```
espresso(F, D) {
    R = 计算 (F ∪ D) 的补集;
    F = 扩展(F, R);
    F = 去冗余(F, D);
    E = 提取相对必要集;
    F = F - E;   D = D ∪ E;
    ...
}
```

```
espresso(F, D) {
    …
    重复执行 {
        φ2 = 当前 F 的蕴含项和布尔文字的加权成本;
        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 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);
    …
}
```

last_gasp函数

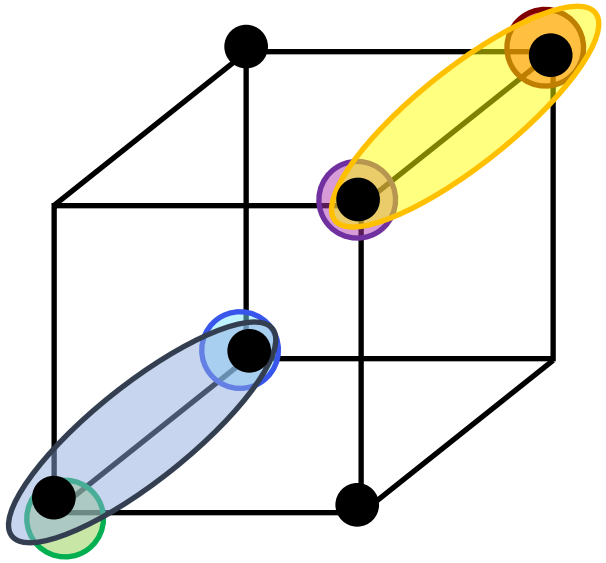
1.独立地对各个蕴含项进行约简（约简顺序无关）

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



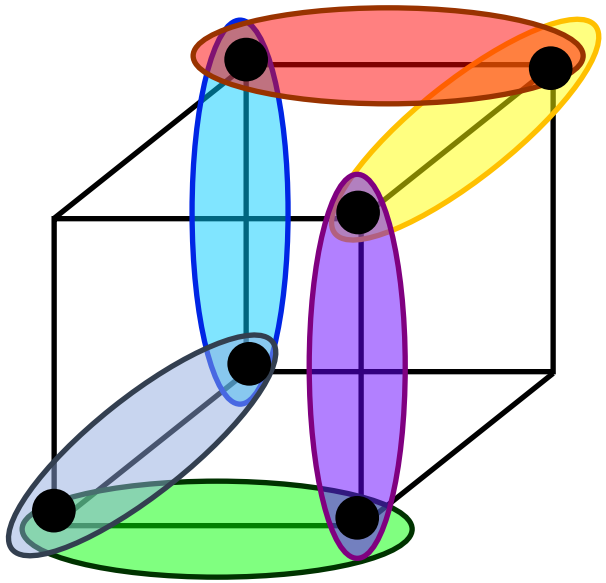
last_gasp函数

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

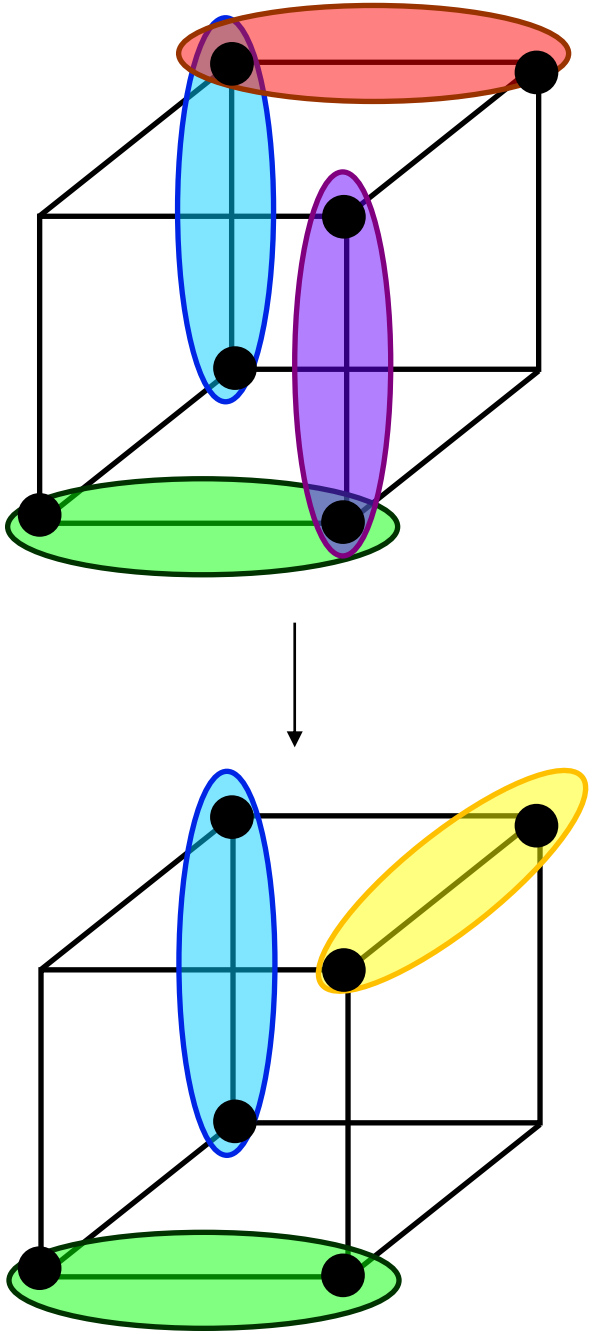


last_gasp函数

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) {
    …
    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
}
```

```
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

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