今日の話題

- VLIWでは、命令スケジューリングはコンパイラが行う
  - 基本ブロック内並列度は限られている
    - 基本ブロック内命令数は5くらいしかない
  - 基本ブロック境界を超えた命令移動
  - 広域命令スケジューリング

- 課題
  - どのような制御パスを通っても正しくなければならない
    - 単一パスのみ考えれば良いわけではない
    - 複雑
  - レジスタ割り当てとの干渉

内容

- レジスタ割り当て
  - レジスタ割り当てとは
  - Chaitin[1981]のアルゴリズム
  - フェーズ順序問題：スケジューリングとの干渉

- 広域命令移動
  - 基本移動

レジスタ割り当て

- 与えられた数以下のレジスタで変数にレジスタを割り当てる

- 不足すれば、スピル・コード挿入

  スピルアウト　：`store 12($sp) = r1`

  再割り当て　　：`r1 = ...`

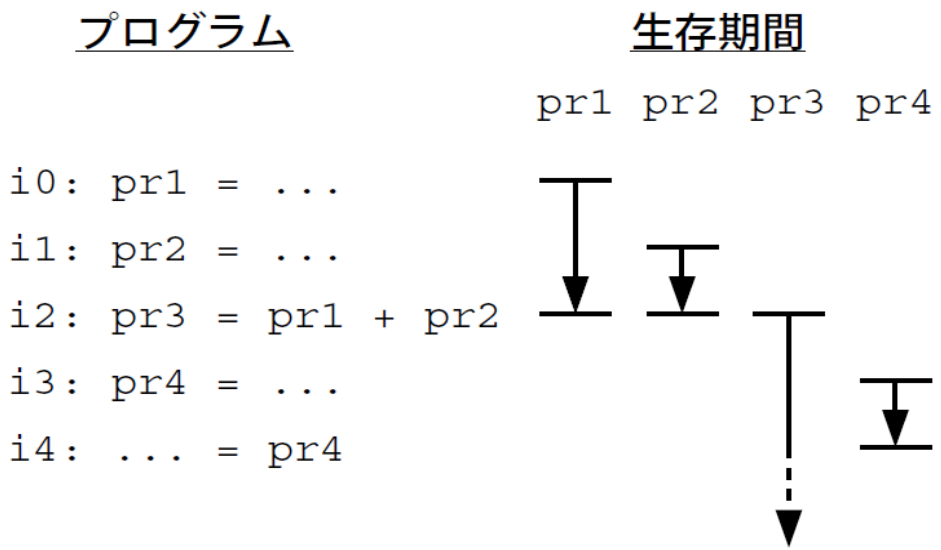
  参照　　：`... = r1`

  スピルイン　　：`load r1 = 12($sp)`

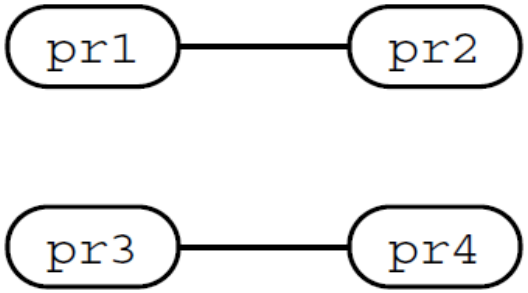
  – 命令数増加による性能低下
    - 実行サイクル数増加
    - メモリ・システム圧迫 → キャッシュ・ミスなど

彩色問題

- 生存期間



- レジスタ干渉グラフ



彩色アルゴリズム(1)

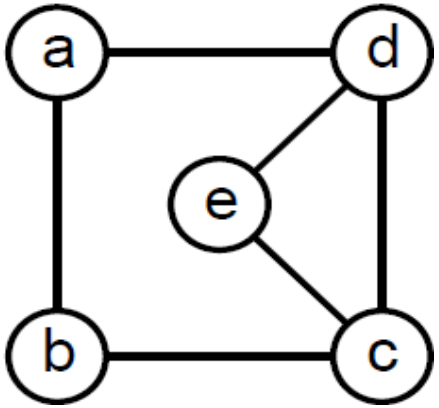
- 彩色問題
  - 与えられた色数でノードを彩色する
  - エッジでつながれたノードは異なる色

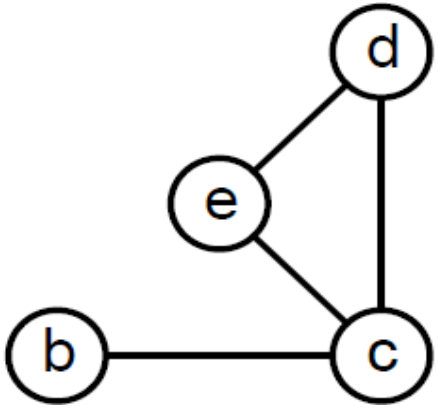
- NP困難

彩色アルゴリズム(2)
―発見的方法―

- Chaitin[1981]のアルゴリズム
  - *G*より色数*k*未満のノードと隣接するノード*n*を見つける
    - *n*と隣接するノードからなるサブグラフ→*k*色彩色可能
    - $G' = G - \{n, e_n\}$が*k*色彩色可能→*G*は*k*色彩色可能
  - $G = G'$として最初に戻る
  - $G = \Phi$ → 手順を逆にたどりノードに色を塗る
  - 途中で彩色不能になったら、適当なノードをスピルアウトする → エッジを除く

例（レジスタ数3）(1)

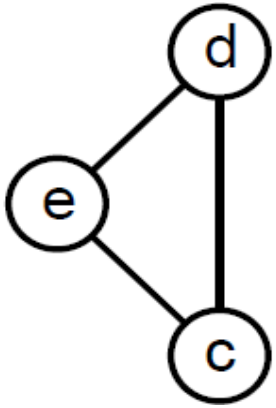




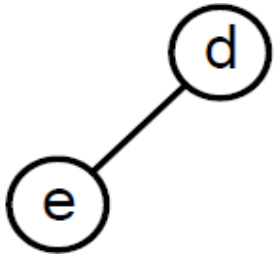
a　b,d

除くノード

除くエッジにつながるノード



a　b,d
b　c



a　b,d
b　c
c　e,d

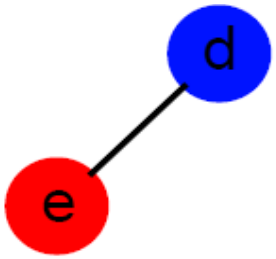
例(2)

a	b,d
b	c
c	e,d
d	e

a	b,d
b	c
c	e,d
d	e
e	

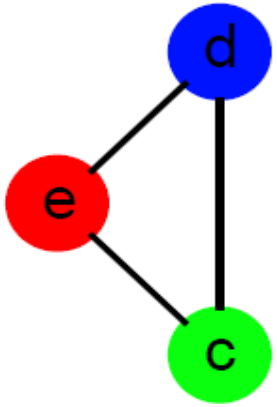
例(3)

a	b,d		
b	c		
c	e,d		
d	e		
e			r1

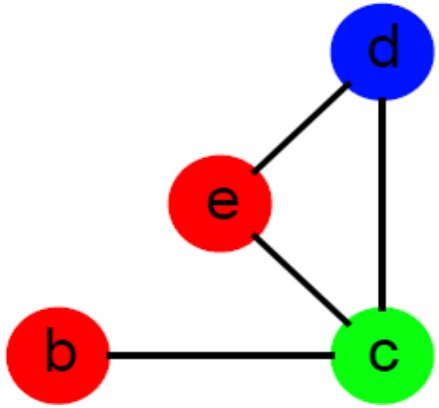


a	b,d		
b	c		
c	e,d		
d	e	r1	r2
e			r1

例(4)

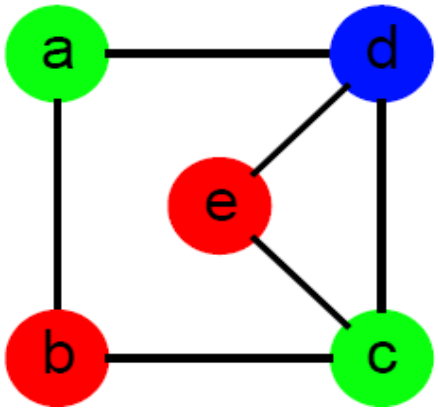


a	b,d		
b	c		
c	e,d	r1,r2	r3
d	e	r1	r2
e			r1



a	b,d		
b	c	r3	r1
c	e,d	r1,r2	r3
d	e	r1	r2
e			r1

例(5)



a	b,d	r1,r2	r3
b	c	r3	r1
c	e,d	r1,r2	r3
d	e	r1	r2
e			r1

内容

- レジスタ割り当て
  - レジスタ割り当てとは
  - Chaitin[1981]のアルゴリズム
  - フェーズ順序問題：スケジューリングとの干渉
- 広域命令移動
  - 基本移動

フェーズ順序問題

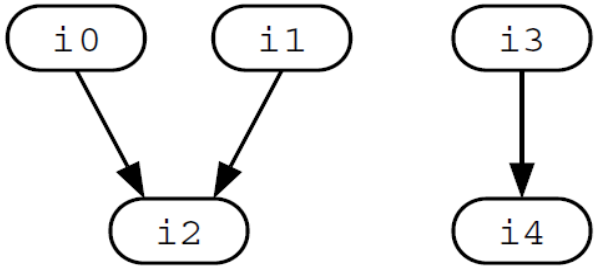
- ポストパス・スケジューリング
  - レジスタ割り当て → スケジューリング
- プリパス・スケジューリング
  - スケジューリング→レジスタ割り当て

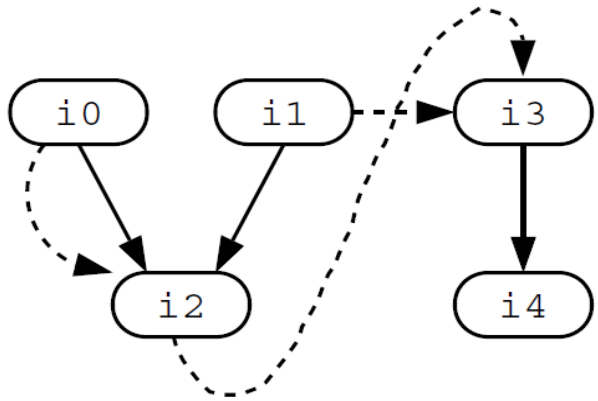
ポストパス・スケジューリング

- レジスタ割り当て → スケジューリング
- 使用レジスタ数をできるだけ少なくする
  → レジスタの再利用
  → 偽の依存(逆依存と出力依存)増加
  → 並列性減少
- 先行制約グラフが高くならないように、レジスタ割り当てを考慮[Goodman 1988]

偽の依存による並列性減少

	レジスタの割り当て前	レジスタの割り当て後
i0	pr1 = ...	r1 = ...
i1	pr2 = ...	r2 = ...
i2	pr3 = pr1 + pr2	r1 = r1 + r2
i3	pr4 = ...	r2 = ...
i4	... = pr4	... = r2





プリパス・スケジューリング

- スケジューリング→レジスタ割り当て

- 並列性最大
  → 生きているレジスタ数増加
  → 必要レジスタ数増加
  → レジスタ不足
  → スピル・コード挿入
  → コード量増大によるメモリ・システム圧迫

- レジスタの生存期間が短くなるようにスケジューリングを考慮
  - 優先度調整[Chang 1995]
    - より多くのソース・レジスタを持つ命令の優先度をあげる
    - デスティネーション・レジスタを持たない命令の優先度を下げる

フェーズ順序問題への別のアプローチ

- 統合[Goodman 1988][Bradlee 1991]
  - スケジューリングしつつレジスタを割り当てる
  - 使用可能なレジスタが多い場合は、並列度を最大化するようスケジューリング
  - そうでない場合は、生存レジスタ数が減少するようスケジューリング

雑談: Stanford大学での宿題(1)

- Goodmanの方法の欠点
  - 並列度とレジスタの不足のバランスの悪化度がある程度小さい場合にはうまくいくが、並列度の方がレジスタ数より非常に大きいときは、うまくいかない
  - レジスタを確保するよう常に頑張らないといけない
    - スピルが多発し、性能低下

雑談：Stanford大学での宿題(2)

- 並列度が非常に大きいプログラムでもうまくいく方法を考案し、実装、評価せよ
  - ベンチマーク：SPEC95 fpppp

- 〆切：2週間後!!!
  - 論文を読んで勉強
  - Goodmanの方法を実装し評価
  - 問題が何なのかを評価し、考える
  - 新しい方法を考案する
  - 実装し評価する
  - 試行錯誤

内容

- レジスタ割り当て

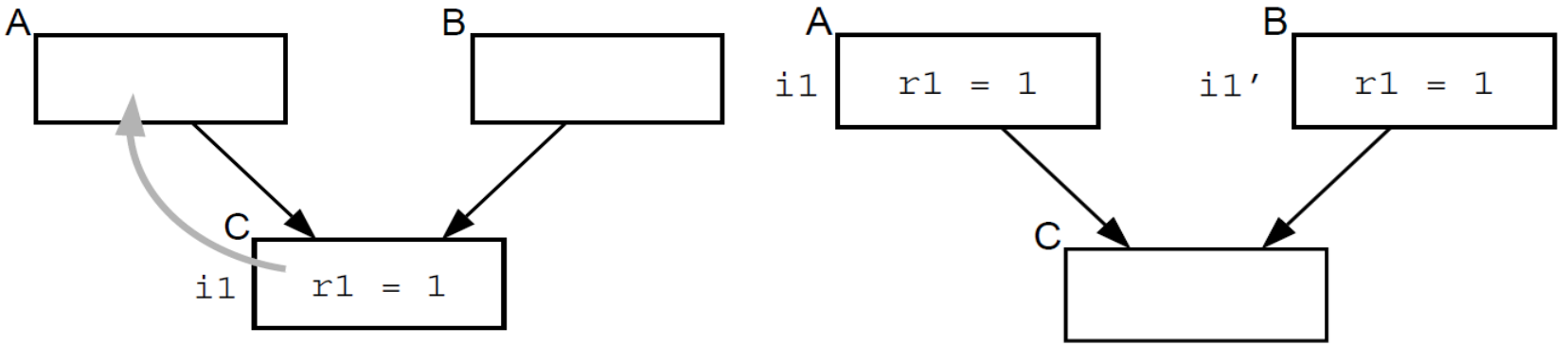
- 広域命令移動
  - 基本移動
  - どのような広域命令スケジューリングもこれの組み合わせ
  - 必ずしもすべてを採用しているわけではない
    - 有効性、複雑度などを考慮

広域命令移動

- 定義
  - 基本ブロック境界を越える命令移動
    - 分岐点を越える上方/下方命令移動
    - 合流点を越える上方/下方命令移動

- 考慮すべきこと
  - プログラムの意味が変わらない
    - 補償コードの挿入：ブックキーピング
  - 有効性：実行サイクル数が減少
    - 簡単にはわからない
    - 例：補償コード → 実行命令数増加、コード量増加

合流点を越える上方命令移動

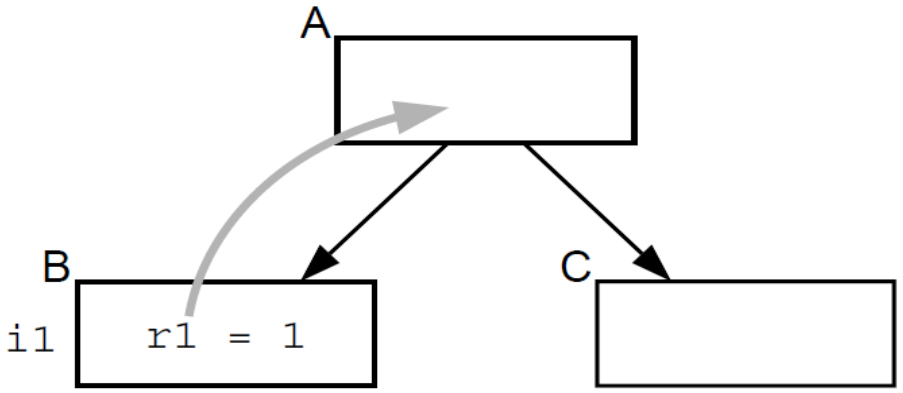


移動前　　移動後

- 複写が必要
- 複写側パスでは有害かもしれない

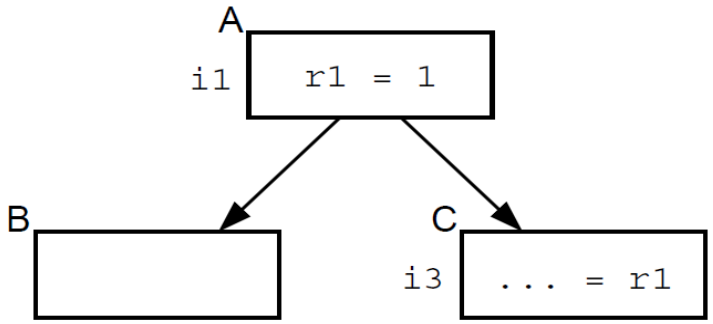
分岐点を越える上方命令移動

- 投機的命令移動
  - 実行されるかどうかは、分岐に依存している
  - 移動後は、分岐の結果に関わらず実行される
  - 一般には、不正
  - 移動パスは、一般には有効
  - 移動パス外では、有害の可能性あり

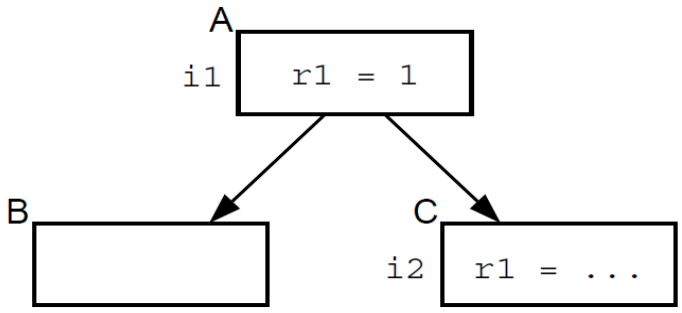


命令移動の正当性

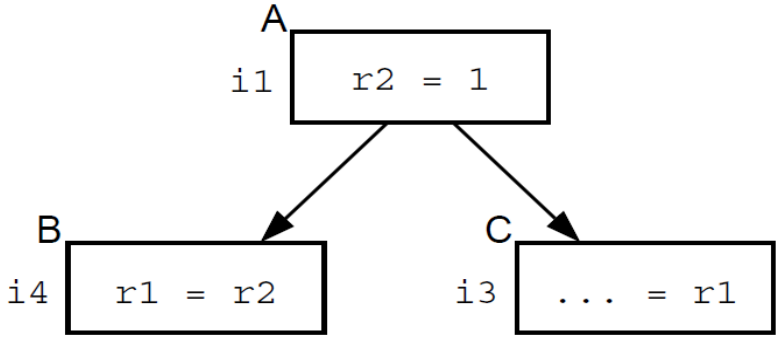
- 移動パスでない方のパスのレジスタ生存による



不正

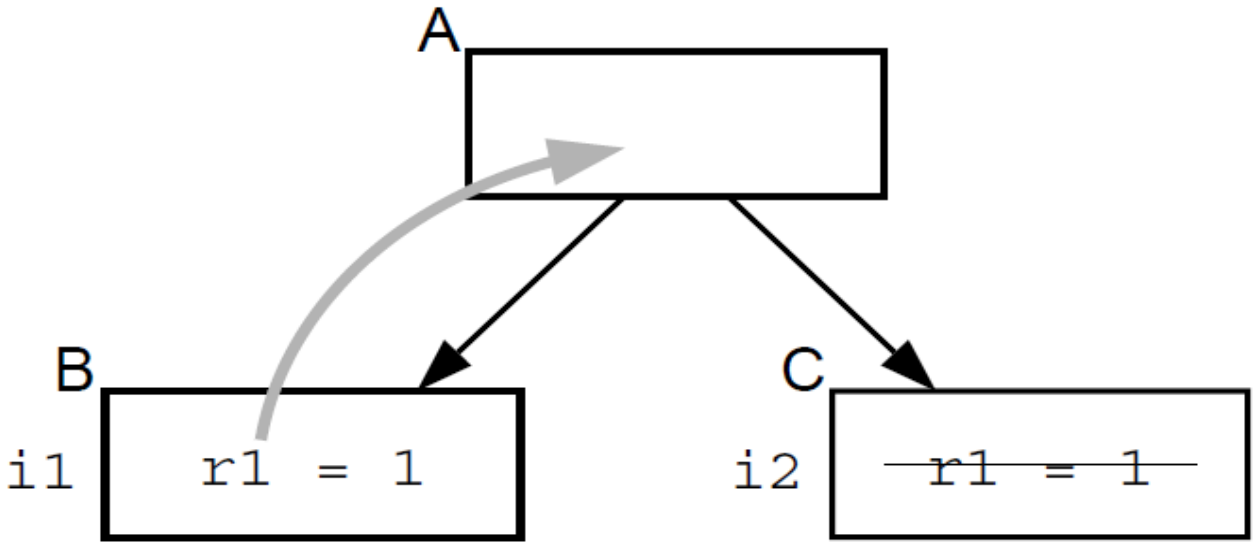


正当



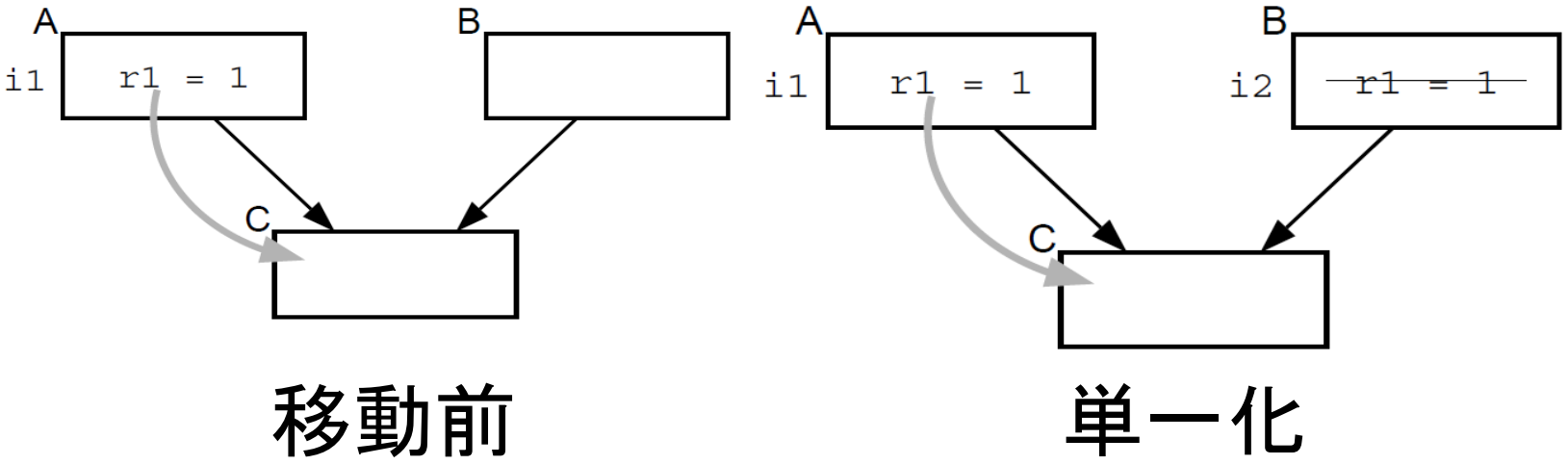
レジスタ・リネーミング

単一化



- 正当、かつ、有効
  - 命令移動を繰り返していると生じうる
  - 合流点越え移動 → 補償コード挿入→分岐点を越え移動 → 単一化

合流点を越える下方命令移動

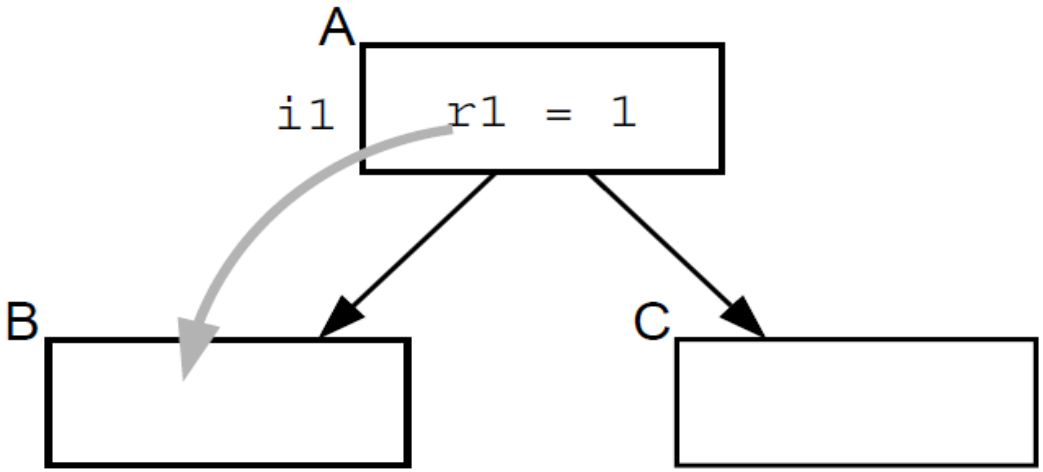


移動前　　単一化

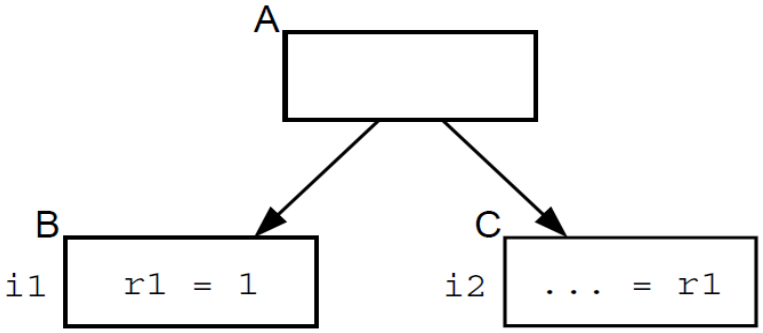
- 一般には、不正
- 単一化できれば、正当

分岐点を越える下方命令移動

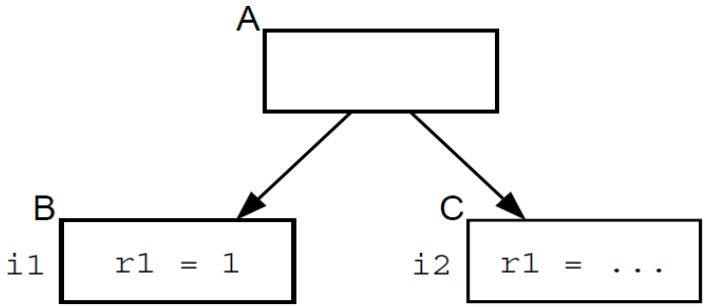
- 一般には、不正
- 移動パス外で使用されていないなら、正当



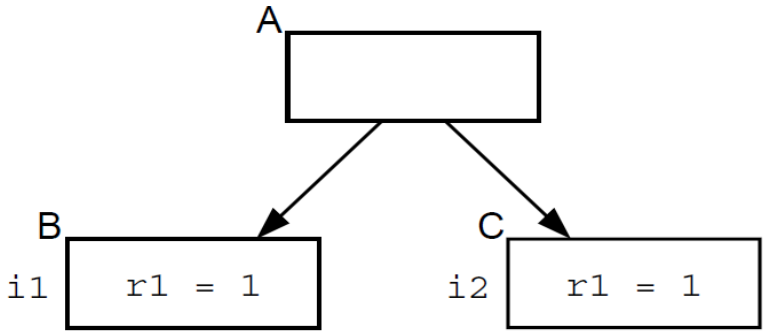
命令移動の正当性



不正



正当



複写

まとめ

- レジスタ割り当て
  - Chaitinのアルゴリズム
    - 彩色問題
  - フェーズ順序問題
    - ポストパス vs. プリパス

- 広域命令移動
  - 基本ブロックを超える命令移動
  - 正当性と有効性
  - 4種の基本移動とブックキーピング
  - どのような広域命令スケジューリングも
    - これを組み合わせている
    - すべて使っているわけではない