ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO di INGEGNERIA DELL'ENERGIA ELETTRICA
E DELL'INFORMAZIONE
"Guglielmo Marconi"
DEI

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE
CLASSE LM 25

TESI DI LAUREA
in
Sistemi di Elaborazione e Controllo

Modelli ILP per la determinazione delle prestazioni conseguibili con il protocollo "Priority Inheritance" in applicazioni multitasking Hard Real-Time

CANDIDATO
Pierfrancesco Ranaldo

RELATORE
Chiar.mo Prof. Eugenio Faldella

Anno Accademico
2015/2016

INDICE

CAPITOLO IV - Relazioni per l'ordinamento temporale nell'accesso alle risorse

CAPITOLO V - Prove sperimentali condotte per i casi di studio analizzati

CAPITOLO I

Fondamenti teorici dei sistemi real-time[1]

1.1 Introduzione ai sistemi real-time

Uno degli argomenti di maggior interesse dell'Ingegneria Informatica è quello dei *sistemi in tempo reale (o "real-time")*. Le loro applicazioni sono infatti presenti in numerosi ambiti come: la robotica, i sistemi di controllo del volo per i veicoli aerospaziali, il controllo di processi produttivi complessi, la regolazione di impianti industriali, le telecomunicazioni, ecc. .

I sistemi real-time sono strettamente legati al concetto di *determinismo*, in quanto il loro compito principale è garantire *il conseguimento di un dato risultato, rispettando le specifiche* imposte dall'applicazione e *riducendo al minimo gli elementi di incertezza.*

In particolare, garantire che un processo svolga il proprio compito *entro una determinata soglia temporale* consente l'accrescimento del livello di *determinismo* del sistema considerato.

1.2 Sistemi real-time e predicibilità

Le peculiarità fondamentali dei sistemi in tempo reale sono due:

1. la correttezza dei risultati di un'elaborazione dipende *anche dal tempo entro cui essi sono prodotti*;
2. il sistema deve essere modellato tenendo ben presente gli *aspetti funzionali* ed *i requisiti temporali* dell'ambiente in cui esso debba poi operare.

Per un sistema, disporre di una *capacità d'elaborazione particolarmente elevata* non costituisce una condizione sufficiente per poter considerare quello stesso come real-time. Questa affermazione è valida perché *risulta ugualmente necessario* dimostrare che il sistema in esame *osservi i vincoli temporali* dell'applicazione di riferimento.

Inoltre, si tenga presente che: se *l'obiettivo di un'elaborazione veloce* è quello di *minimizzare il tempo medio di risposta* dei processi, lo *scopo principale* invece di *un'elaborazione real-time è il soddisfacimento di requisiti temporali individuali dei task*.

In conclusione, l'aspetto più importante che debba possedere un SRT (sistema real-time) è la *predicibilità*, e cioè: *il poter garantire la terminazione di ogni task da esso gestito, in un tempo predeterminato*.

---

1 Tutti gli argomenti discussi in questo capitolo sono una semplice rielaborazione riassuntiva di quanto già noto nell'ambito dei Sistemi in Tempo Reale. L'attuale Capitolo I non è frutto di idee originali del laureando, ed è altresì interamente tratto dai testi e dai documenti [1], [2], [3], [4], [5], [6], [7] elencati dettagliatamente nella sezione bibliografica presente alla fine di questo elaborato.

È possibile *misurare la predicibilità di un SRT*, cioè *il livello di rispetto dei requisiti temporali offerto per i processi da esso gestiti*.

Secondo tale misura si può classificare i SRT come:

- *con predicibilità deterministica,* cioè garanti del limite di tempo per ogni processo;
- *con predicibilità probabilistica,* tali che *la probabilità della predicibilità* di un generico processo non sia inferiore ad un dato valore [0;1];
- ed infine, con *predicibilità deterministica a run-time*, ovvero: solo al momento dell'attivazione del task è possibile stabilire se il sistema sia o meno in grado di rispettare la sua terminazione secondo un tempo prefissato.

Per un SRT, il livello di predicibilità è influenzato da numerosissimi e svariati fattori come l'architettura fisica dell'elaboratore, i meccanismi di gestione della memoria, le primitive del core del sistema operativo, le tecniche DMA, i linguaggi di programmazione, ecc. .

1.3 Processi real-time e principali caratteristiche

L'aspetto di primaria importanza che contraddistingue un *processo real-time* è che esso possiede una *deadline*, e cioè *un limite temporale entro il quale la propria esecuzione debba essere terminata*. In realtà, *l'importanza* assunta dalle deadline dei task di un'applicazione real-time può essere differente. Questo permette di classificare i processi r-t come:

a) *processi hard r-t. L'osservazione del loro limite temporale è di primaria importanza*, e se ciò non avvenisse si andrebbe incontro a conseguenze disastrose (vedi caso di un sistema di allarmi di una centrale nucleare);
b) *processi soft r-t.* Per essi, *in alcuni casi specifici, è possibile trascurare il rispetto dei vincoli temporali* (vedi caso del sistema di posizionamento di una nave).

Un'ulteriore categorizzazione dei p. r-t (processi real-time) è relativa alla *frequenza con cui di essi è richiesta l'esecuzione.* In particolare, si parla di:

1. *processi periodici*, per quelli *innescati ad intervalli regolari e costanti di tempo*;
2. *processi aperiodici* (*p. hard r-t*) e *sporadici* (*p. soft r-t*), se invece non possiedono una *frequenza costante* rispetto alla quale sono *attivati*.

In realtà più processi r-t costituiscono assieme *un'applicazione real-time* se, oltre ad essere indicato *l'insieme dei parametri temporali di ciascun task*, sono specificate anche: i loro livelli di priorità, la *strategia di pianificazione della loro esecuzione* e *le politiche di gestione dell'accesso a risorse condivise*.

Riferendosi ora alla *i-esima istanza di un processo r-t*, chiamata *job*, *l'insieme dei parametri temporali di un task* include:

- l'*arrival time* a_i;
- lo *start time* s_i;
- il *finish time* f_i;
- la *deadline* d_i;

da cui è semplice ricavare:

- il *computation time* $C_i = f_i - s_i$;
- la *relative deadline* $D_i = d_i - a_i$;
- il *response time* $R_i = f_i - a_i$;
- la *lateness* $L_i = f_i - d_i$;
- l'*exceeding time* (anche noto come "*tardiness*"), $E_i = \max(0, L_i) =_{def}$ il tempo di completamento del task *qualora non fosse rispettata la deadline* ;
- lo *slack time* (anche noto come *"laxicity"*) $X_i = D_i - C_i$, definito come *il tempo disponibile al completamento del job rispettando la deadline*.

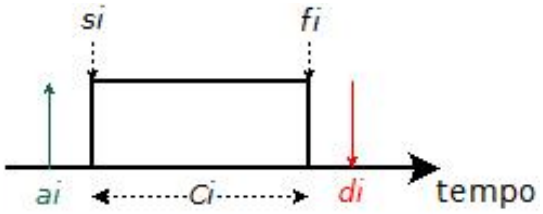


Figura 1.1 Principali parametri temporali di un processo real-time

L'elenco delle caratterizzazioni temporali appena fatta può essere descritta meglio notando come:

1. per i processi periodici, prendendo in considerazione *i job i-esimo* e quello successivo *(i+1)-esimo*, allora: il *periodo T del task* è per definizione la *differenza tra gli arrival time*, e cioè $T =_{def} a_{i+1} - a_i$, nonché *coincide con la deadline relativa* D_i di ciascun job_i, ovvero $D_i =_{def} T$. Infine, per *fase* Φ del task periodico si intende l'arrival time a_1 della sua prima istanza.
2. per i processi aperiodici/sporadici, è necessariamente presente anche il *MIT (Minimum Interarrival Time)*, che rappresenta *la distanza temporale media tra gli arrival time di due job (i) e (i+1) successivi.* È facile comprendere pertanto come: $\forall i$, $a_{i+1} - a_i \geq MIT > D_i$.

1.4 Funzione di utilità dei processi real-time

Come detto in precedenza, la correttezza dell'elaborazione svolta da parte di un processo r-t dipende strettamente anche dal rispetto della deadline associata. È possibile conseguentemente fare una *misura dell'utilità del risultato prodotto da un processo*, a seconda o meno che esso sia r-t e che sia *completato in tempo.*

A tal proposito si adopera la *funzione di utilità $v(f_i)$, dipendente dal finish time f_i del job i-esimo del task in esame.* È possibile osservare come:

1. L'utilità di un processo *non real-time* è costante, e prescinde effettivamente dal finish time.

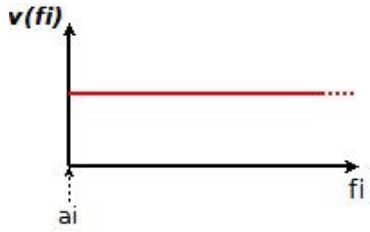


Figura 1.2: $v(f_i)$ per un processo non r-t

2. L'utilità per un task *soft real-time* decresce rapidamente una volta superata la deadline.

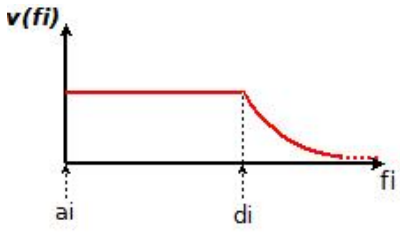


Figura 1.3: $v(f_i)$ per un processo soft r-t

3. L'utilità per un task *hard real-time è nulla* qualora non sia rispettata la deadline.

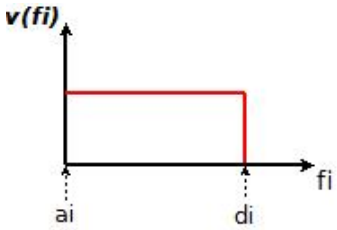


Figura 1.4: $v(f_i)$ per un processo hard r-t

Un'ulteriore precisazione riguarda i processi hard r-t il cui completamento *dopo la propria deadline risulta addirittura dannoso* per il sistema. In tal caso la *funzione di utilità* $v(f_i)$ *tende al valore meno infinito per* d_i.

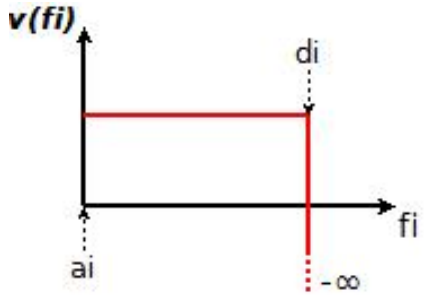


Figura 1.5: $v(f_i)$ per un task hard r-t interrotto se risulti violata la propria deadline

1.5 Strategie di scheduling per processi real-time

Nei sistemi r-t riveste un ruolo di primaria importanza *la pianificazione dell'esecuzione dei processi*, o più semplicemente la *strategia di scheduling*. Essa stabilisce quale sia la modalità/l'ordine con cui sono eseguiti i task dell'applicazione di modo che siano osservate le relative deadline. Il componente software che in un sistema operativo *attua la politica di scheduling* prende il nome di *scheduler.*

Il modo più semplice per definire una *politica di schedulazione* consiste nell'utilizzo di diagrammi temporali, nei quali lungo un asse temporale sono disposte opportunamente i tempi di esecuzione dei vari processi, cercando di rispettare i requisiti temporali. Questa tecnica prende il nome di *schedulazione ad hoc*.
Tuttavia essa non rappresenta la *miglior scelta*, in quanto l'individuazione di una *pianificazione* costituisce generalmente un *problema non banale.* Inotre, è sufficiente anche solo una piccola modifica delle caratteristiche di uno dei task dell'applicazione per rendere vano lo sforzo computazionale fatto in precedenza per l'individuazione della *politica di scheduling ad hoc.*

Anche per le strategie di scheduling esistono varie modalità con cui classificarle, e cioè:

- a seconda che esse siano stabilite a *tempo di esecuzione* ($=_{def}$ *s. è di tipo online)*, o meno ($=_{def}$ *s. tipo offline);*
- in funzione del fatto che *i parametri dei task possano mutare* ($=_{def}$ *s. di tipo dinamico)*, oppure no ($=_{def}$ *s.di tipo statico)*;

- sulla base che si *cerchi di rispettare i requisiti temporali di tutti processi dell'applicazione (=$_{def}$ s. di tipo guaranteed)*, o esclusivamente di *ottimizzare le prestazioni medie dei task (=$_{def}$ s. di tipo best effort)*.
- ed infine, *se esse consentano o meno l'interruzione temporanea dei processi dell'applicazione* (=$_{def}$ *s. di tipo preemptive/non-preemptive*).

In realtà, è molto frequente che i processi r-t di un'applicazione *possiedano differenti livelli di priorità*, quindi la presenza di vincoli di precedenza limita ulteriormente le possibili scelte nel momento della definizione di una politica di schedulazione. Questo problema diviene ancora più complesso se in aggiunta è presente un set di risorse condivise, il cui accesso è regolato da meccanismi di mutua esclusione e/o meccanismi di sincronizzazione vari.

1.6 Politiche di schedulazione per processi r-t periodici

Le *strategie di pianificazione dei processi r-t periodici* possono essere inquadrate all'interno di due principali categorie:

- politiche di s. *clock-driven,* ovvero pianificazioni di tipo *offline, guaranteed e non--preemptive,* attraverso le quali si stabilisce in ogni istante temporale quale sia il processo che debba essere eseguito. Queste strategie sono particolarmente efficienti, in quanto è possibile individuare la pianificazione più adatta all'applicazione r-t in esame, nonché il sistema risente esclusivamente dell'overhead associato al context-switch dei processi e della presenza di meccanismi di comunicazione tra essi. Tuttavia, queste politiche sono poco flessibili circa l'alterazione dei parametri dei task ed in relazione all'eventuale introduzione di nuovi processi o di task aperiodici.
- politiche di s. *priority-driven*, cioè strategie di tipo *online e preemptive*, che in generale *minimizzano gli intervalli temporali di non utilizzo* della CPU. Per queste pianificazioni, *la scelta del task da mandare in esecuzione avviene sulla base di un livello di priorità* attribuito *staticamente* o *dinamicamente.* Inoltre, in virtù del fatto che un processo possa essere *sospeso*, allora per ciascuno di essi è introdotto uno *stato "di idle",* uno "*di ready"* ed uno "*di running"*.

1.7 Principali politiche di schedulazione priority-driven

Nell'ambito delle strategie di pianificazione *di tipo priority-driven per processi periodici*, assumono maggiore rilievo:

1. *la schedulazione Rate Monotonic Priority Order (RMPO)*. Essa prevede l'attribuzione *statica a ciascun processo P_i di un livello di priorità pr_i inversamente proporzionale al proprio periodo T_i.* Inoltre *questo algoritmo di pianificazione è ottimo,* e cioè: se un'applicazione è *schedulabile attraverso una qualsiasi strategia che preveda l'attribuzione statica dei livelli di priorità*, allora lo è anche con RMPO. Vale anche il viceversa relativamente in caso di "*non schedulabilità"* delle applicazioni;
2. *la schedulazione Deadline Monotonic Priority Order (DMPO).* Questa strategia presenta le medesime caratteristiche di RMPO, e si differenzia altresì esclusivamente per il fatto che la priorità pr_i di ciascun task P_i è inversamente proporzionale alla propria deadline relativa D_i (si noti come RMPO e DMPO coincidano se: $T_i = D_i$). Anche DMPO costituisce una tecnica di pianificazione *ottima;*
3. *la schedulazione Earliest Deadline First (EDF).* Tale algoritmo assegna *dinamicamente* i livelli di priorità ai task dell'applicazione in funzione di quale sia quello più prossimo alla violazione della propria deadline.

1.8 Condivisione delle risorse tra processi real-time e problematiche connesse

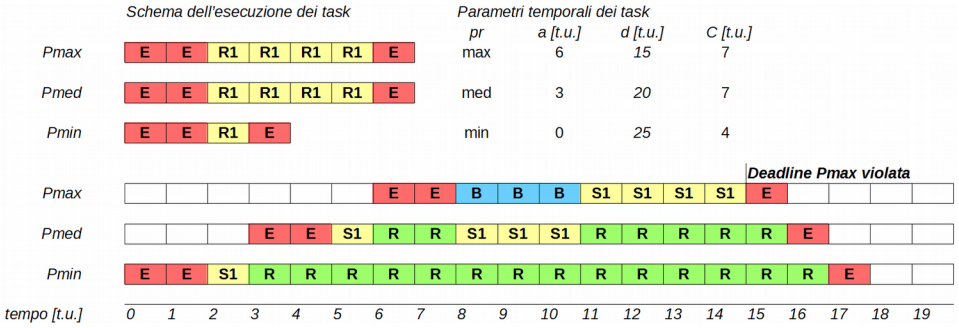
La presenza di un insieme di risorse condivise in un'applicazione r-t costituisce uno scenario che è possibile incontrare frequentemente. Ciò aumenta notevolmente *il grado di complessità con cui poter pianificare l'esecuzione dei processi*, in quanto l'accesso a ciascuna di esse è generalmente *mutuamente esclusivo* ed i processi richiedono il loro utilizzo *in maniera concorrente*.

In questo contesto, un primo problema è quello dell'*inversione di priorità*. A tal proposito, si consideri il caso di un task set costituito da tre processi $\{P_{MAX}, P_{MED}, P_{MIN}\}$, cui sono attribuiti staticamente i livelli di priorità $pr_{max} > pr_{med} > pr_{min}$. Si supponga inoltre che essi richiedano tutti l'accesso alla risorsa R_1, e che *il possesso* del lock del semaforo corrispondente sia *mutuamente esclusivo*.

Se il lock del semaforo S_1 per la gestione degli accessi di R_1 fosse posseduto da P_{MED} o P_{MIN}, e fosse lanciato il processo P_{MAX}, la *presenza di livelli di priorità sarebbe del tutto inutile* in quanto resterebbe in esecuzione il task $P_i \in \{P_{MED}, P_{MIN}\}$ con $pr_i < pr_{max}$ sino al termine della propria sezione critica con utilizzo di R_1.

Questo fenomeno è deprecabile perché è evidente che *processi a minor priorità possono causare blocchi di esecuzione*, con possibili *violazioni dei requisiti temporali, per quelli a più elevata priorità*.

Nel diagramma temporale di pagina seguente è mostrato graficamente quanto appena descritto.



notazione:

E := {il processo è in esecuzione};

S_1 := {il processo *detiene il lock di* S_1 *ed è in esecuzione di una propria sezione critica*};

R := {il processo è *temporaneamente sospeso da uno a priorità superiore*};

B := {il processo è *bloccato da un task a priorità inferiore*}

Figura 1.6 Esempio per il fenomeno dell'inversione di priorità

La figura di sopra mette in evidenza come *per il processo* P_{MAX} *non sia rispettata la relativa deadline*.

Il fenomeno dell'inversione di priorità ha effetti maggiormente dannosi se ad esempio il task P_{MED} non possedesse alcuna sezione critica, ed al contempo quella di P_{MIN} fosse temporalmente più estesa rispetto al caso appena esaminato.

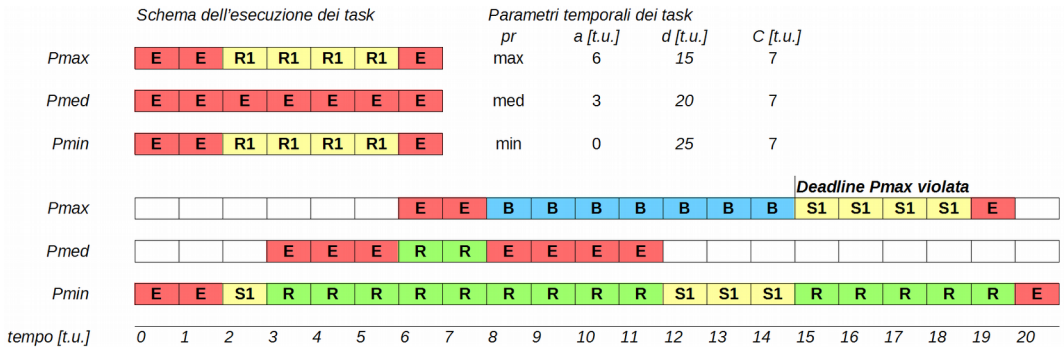


Figura 1.7 Esempio per il fenomeno dell'inversione di priorità *incontrollata*

La diagramma di sopra illustra quanto appena detto. In particolare, in questo caso è possibile osservare come:

a) il lock del semaforo di R_1 sia detenuto da P_{MIN}, il quale causa il blocco del processo P_{MAX};

b) il task P_{MED}, essendo a priorità superiore rispetto a P_{MIN} e non richiedendo alcuna risorsa, entra nuovamente esecuzione sino a che non risulti terminato.

In conclusione: il processo P_{MAX} subisce *un blocco complessivamente definito da P_{MED} e P_{MIN}*. In questo caso si parla di *inversione di priorità incontrollata*.
Un semplice accorgimento per limitare in parte questo problema, potrebbe essere quello di *sospendere il meccanismo di preemption* qualora il task in esecuzione *sia all'interno di una propria sezione critica* (*$=_{def}$ il set di istruzioni con utilizzo di una risorsa condivisa*). Questa soluzione è tanto semplice quanto poco efficace per gli scenari relativi a task set in cui i processi a minor priorità, *accedendo a risorse non richieste dai processi "più critici"*, causano ugualmente il blocco dei task a maggiore priorità.

Un altro problema legato alla gestione degli accessi concorrenti alle risorse condivise è quello della *concatenazione di blocchi,* e cioè: un processo, in corrispondenza dell'esecuzione di ogni propria sezione critica, può incorrere in un blocco da parte di un task a priorità inferiore. L'immagine seguente mostra come P_{MAX} preveda l'esecuzione di due sezioni critiche con utilizzo delle risorse R_2 e R_1, ma i lock dei relativi semafori S_2 e S_1 siano rispettivamente detenuti da P_{MED} e P_{MIN}.
In questa situazione P_{MAX} è arrestato da entrambi i processi con priorità inferiore, è ciò determina l'infrazione della relativa deadline.

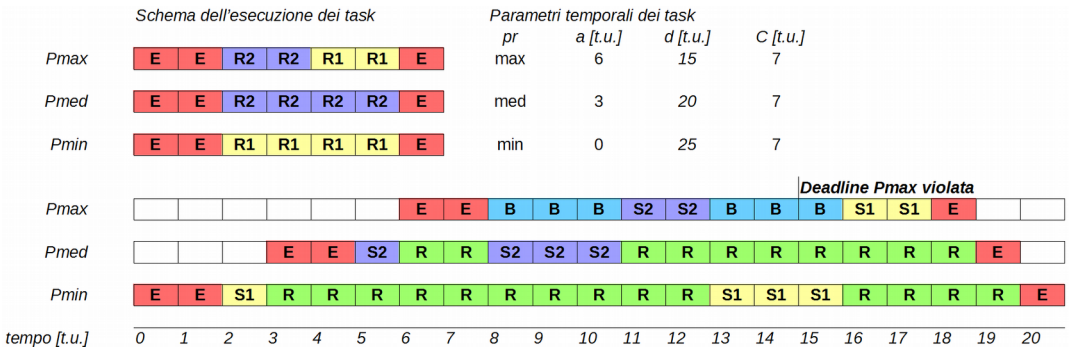


Figura 1.8 Esempio per il fenomeno della *concatenazione di blocchi*

Un ultimo fenomeno problematico è *il deadlock*. Esso si verifica quando un processo P_i con priorità pr_i detiene il lock di una generica risorsa R_a del Resource Set e richiede l'accesso ad un'ulteriore risorsa R_b, il cui lock del relativo semaforo è però in possesso di un task P_k a priorità inferiore. Tuttavia P_k, per poter proseguire la propria esecuzione ha necessità di utilizzare R_a, e

ciò implica che *entrambi i processi rimangano bloccati per un tempo indefinito.* Un esempio di questa tipologia di problema è rappresentata nella figura 1.9 successiva, nell'ipotesi che il task set sia costituito dai processi $\{P_{MAX},P_{MIN}\}$ con $pr_{MAX} > pr_{MIN}$, ed il Resource Set includa le risorse $\{R_1, R_2\}$.

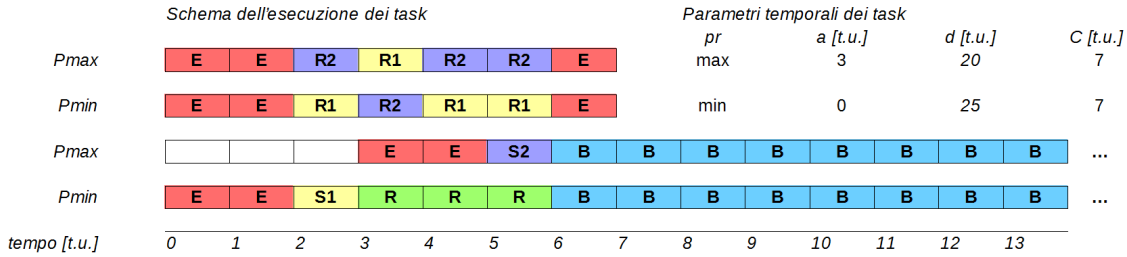


Figura 1.9 Esempio per il fenomeno del *deadlock*

Osservando la figura si capisce come in corrispondenza dell'istante temporale *in cui* P_{MAX} *richieda l'accesso a* R_1, *entrambi i task cadano in una condizione di stallo*.
Il *problema del deadlock* è in realtà quello meno rilevante poiché è così diffuso nel contesto dell'Informatica, che risultano disponibili in letteratura numerose tecniche per rilevare e risolvere tali *situazioni pericolose*.

1.9 Introduzione al protocollo Priority Inheritance

L'analisi dei precedenti esempi fa emergere la necessità di utilizzare adeguati *protocolli per la gestione degli accessi alle risorse condivise*, tali da minimizzare la frequenza con cui i task possano causare blocchi vicendevoli in un'applicazione real-time.
Il protocollo Priority Inheritance (PI) è quello preso in riferimento nello sviluppo di questo lavoro di tesi in virtù della sua estrema diffusione nel mondo dei sistemi operativi r-t, quindi l'obiettivo dell'attuale e dei seguenti paragrafi è esaminare dettagliatamente i suoi principali aspetti.
Prima di procedere con l'analisi di PI, è necessario introdurre le seguenti ipotesi utili alla descrizione del contesto in cui si suppone operi:

- il sistema di elaborazione *possiede una sola CPU*;
- il *Task Set* dell'applicazione è costituito da N processi $\{P_1, P_2, \dots P_n\}$, ed a ciascuno di essi *è assegnata staticamente una priorità* $p(P_i)$ al momento della loro creazione. Esse sono sono possedute anche dalle corrispondenti istanze;
- *l'ordine* delle priorità è: $p(P_1) > p(P_2) > p(P_3) > \dots > p(P_n)$;

- *un job è un'istanza di un task dell'applicazione, costituita da una sequenza di istruzioni che è eseguita dal processore senza interruzioni sino al suo completamento;*
- un job *è bloccato* se la sua esecuzione è *interrotta temporaneamente* da un altro job con *priorità inferiore;*
- è fondamentale che *ciascun job completi la sua esecuzione prima che giunga l'istanza successiva*;
- un *task periodico* è definito come la sequenza *di job identici* rilasciati *ad intervalli regolari di tempo.* Si parla invece di *task aperiodico* quando ci si riferisce al caso in cui le relative istanze giungano nel sistema *ad intervalli irregolari di tempo*;
- la *strategia di schedulazione* adottata è *RMPO* o *DMPO;*
- se *due job sono pronti* per essere eseguiti, allora *il processore è assegnato a quello con priorità più elevata*. Inoltre, qualora i job *in stato di ready* abbiano *stessa priorità*, allora essi saranno serviti secondo *la politica First-Come-First-Served*;
- il *Resource Set* include le risorse $\{R_1, R_2, \ldots, R_m\}$, e per ognuna di esse *è disponibile una sola unità,* il cui *accesso è regolato attraverso un semaforo binario.*

1.10 Notazione per l'analisi del protocollo Priority Inheritance

La notazione adoperata nelle sezioni successive è:

- J_i rappresenta l'i-esimo job del processo P_i incluso nel Task Set dell'applicazione;
- $z_{i,k}$ indica la k-esima *sezione critica* di J_i;
- $S_{i,k}$ è il semaforo binario che gestisce la risorsa utilizzata in $z_{i,k}$;
- $P(S_{i,k})$ e $V(S_{i,k})$ rappresentano *le operazioni indivisibili di wait* e *signal sul semaforo* $S_{i,k}$.
- Si ricordi che una *sezione critica* è definita come la sequenza di istruzioni di un job che risulta racchiusa tra un'operazione di wait P(S) e la corrispondente operazione di signal V(S).

Si suppone inoltre che un generico job:

- possa avere *sezioni critiche non annidate tra loro* del tipo $\{\ldots P(S_{i,j}) \ldots V(S_{i,j}) \ldots P(S_{i,k}) \ldots V(S_{i,k}) \ldots\}$, tali che: $z_{i,j} \cap z_{i,k} = 0$;
- possa avere *sezioni critiche innestate tra loro* del tipo $\{\ldots P(S_{i,j}) \ldots P(S_{i,k}) \ldots V(S_{i,k}) \ldots V(S_{i,j}) \ldots\}$, tali che: $z_{i,k} \subset z_{i,j}$.

Inoltre per l'analisi del protocollo priority inheritance, si indichi con $\beta_{i,j}$ *l'insieme costituito dalle sezioni critiche* $z_{j,k}$ *di* J_j *che risultano bloccanti per* J_i, quindi $p(J_i) > p(J_j)$, e che β_{ij}* sia l'insieme definito come: $\{z_{j,k} \mid J_j$ *blocca* J_i *e non esiste* $z_{j,i} \in \beta_{i,j}$ *che includa* $z_{j,k}\}$, cioè delle più *estese sezioni critiche di* J_j *che possano causare un arresto temporaneo di* J_i.

1.11 Protocollo Priority Inheritance: un concetto di base

Il principio alla base del protocollo di priority inheritance è il seguente:

- quando un job di priorità bassa possiede una risorsa condivisa e, eseguendo una sua sezione critica, *blocca* uno o più job aventi priorità più elevata, allora *al processo bloccante con priorità inferiore è assegnata temporaneamente il livello di priorità maggiore tra quelli associati ai processi bloccati*. Terminata l'esecuzione della sezione critica, al processo *"bloccante"* sarà nuovamente assegnato il *suo livello di priorità nominale*.

Nel caso dell'esempio precedente mostrato in figura 1.7:

- il job J_3 (instanza di P_3), che è nella sua sezione critica, nonché possiede la risorsa richiesta per l'esecuzione di J_1, con il supporto di PI eredita dal processo bloccato il livello di priorità corrispondente $p(J_1)$, che è il più elevato nel task set considerato.
  In questo caso quindi, il rilascio ora di J_2 non determina né la sospensione di J_3, né una forma di *preemption indiretta subita da J_1* da parte dello stesso J_2.
  Infine, terminata la sezione critica di J_3, siccome sono *ristabiliti i livelli nominali di priorità*, allora andranno in esecuzione: J_1, quindi J_2 ed in ultimo J_3. Rispettando quindi la relazione $p(J_1) \geq p(J_2) \geq p(J_3)$. Nella figura 1.10 è mostrato graficamente quanto appena descritto.

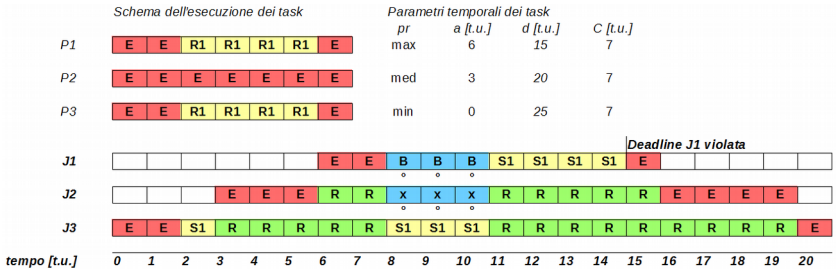


Figura 1.10 esempio #1 di applicazione del protocollo PI ad un task set schedulato con RMPO

Inoltre se π_3 indica la *priorità corrente del job J_3*, allora è possibile osservare come nell'intervallo *[8;11[t.u.* esso coincida con $p(J_1)$ come mostrato nella figura successiva.

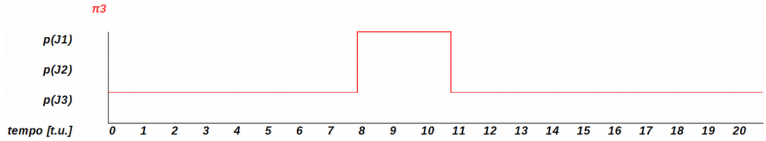


Figura 1.11 andamento temporale della priorità corrente π_3 di J_3 dell'esempio #1

Un'importante osservazione è:

- se la risorsa posseduta da J_3 al momento dell'esecuzione della sua sezione critica non fosse condivisa con J_1, ma con J_2, allora J_3 *erediterebbe il livello di priorità* $p(J_2)$. Quindi: al rilascio di J_1, J_3 sarebbe sospeso, ed andrebbe in esecuzione per l'appunto il job a più alta priorità (ovvero J_1).

  Questo è un comportamento perfettamente coerente con i livelli di priorità stabiliti.

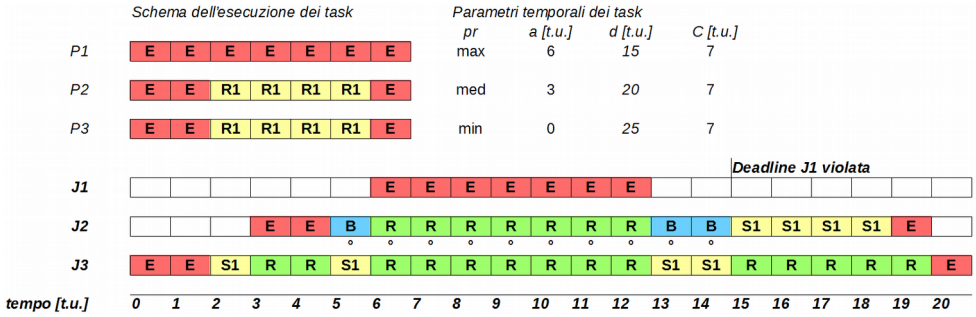


Figura 1.12 esempio #2 di applicazione del protocollo PI ad un task set schedulato con RMPO

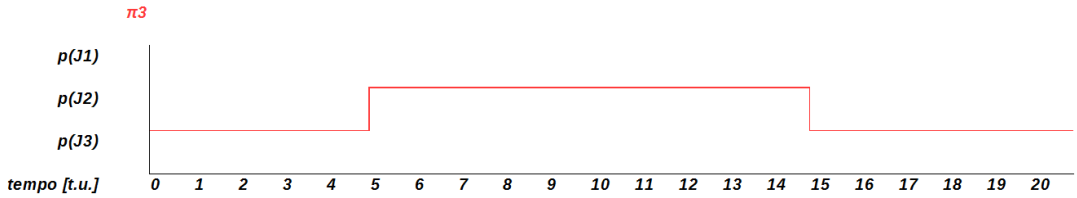


Figura 1.13 andamento temporale della priorità corrente π_3 di J_3 dell'esempio #2

In conclusione, quindi: *il protocollo di sincronizzazione Θ è detto di priority inheritance* se dato un job J_k avente priorità inferiore a quelle dei job J_1, J_2, J_3,...,J_i , eseguendo una propria sezione critica che blocca i job con priorità maggiore, *eredita allora temporaneamente la priorità* $p = max\{p(J_1), p(J_2), ..., p(J_i)\}$.

1.12 Protocollo Priority Inheritance

I fondamenti su cui si basa il protocollo di P.I. sono:

1. Si supponga che il job J è quello a più alta priorità tra quelli *in stato di ready.*

   Ad esso è quindi è assegnato il processore. J necessita di eseguire una sezione critica, nonché richiede l'attribuzione del semaforo S. I possibili scenari sono due, ovvero: se S è già stato assegnato, allora l'accesso ad S da parte di J è negato, nonché J risulta *bloccato*.

   Se al contrario S non è già occupato, e J riesce ad acquisirne conseguentemente il possesso, allora: J esegue la sua sezione critica, terminata la quale rilascia

immediatamente S, così da consentire ad *eventuali job bloccati su S*, di poter essere *risvegliati*.

2. Il job J utilizza il proprio livello di priorità p(J), a meno che nell'esecuzione della sua sezione critica blocca uno o più job a priorità più elevata, tra i quali quello a *priorità maggiore* è J_h con priorità $p(J_h)$. In questo caso: J eredita ed utilizza il livello di priorità $p(J_h)$, fin tanto che si trova nella sua sezione critica. Alla fine di essa, infatti, a J è nuovamente attribuito il suo livello nominale di priorità p(J).
3. In questo protocollo *l'ereditarietà della priorità è transitiva*. E cioè: si considerino i job J_1, J_2 e J_3 con $p(J_1)>p(J_2)>p(J_3)$. Se J_2 è bloccato da J_3, allora quest'ultimo eredita il livello di priorità $p(J_2)$. Ma se a questo punto, se J_2 bloccasse a sua volta J_1, allora il job J_3 erediterebbe $p(J_1)$, per mezzo di J_2. Ciò può avvenire *solo in presenza di sezioni critiche annidate*.
4. Se il job J non è bloccato da J_L, e la priorità di J è superiore a quella assegnata o *ereditata* da J_L, allora J può esercitare la preemption su J_L.

Da quanto appena detto è facile comprendere come i blocchi subiti dai job possano essere di due tipi:

- *Blocco Diretto*. Si verifica quando un processo a priorità elevata è difatti bloccato da uno a priorità inferiore, dato che ne possiede il semaforo della risorsa richiesta dal job più critico.

  Questo tipo di blocco consente di preservare la *consistenza di dati condivisi.*
- *Blocco Push-Through*. Si verifica quando J_M, considerato come *job a priorità intermedia*, è bloccato da J_L, *job a priorità più bassa*, dato che quest'ultimo ha ereditato un livello di priorità più elevato $p(J_H)$ rispetto a $p(J_M)$. Questo tipo di blocco è fondamentale per evitare che J_H subisca una forma di *preemption indiretta* da parte di J_M.

1.13 Proprietà del protocollo PI

Lemma 1-1: un job J_H è bloccato da un altro job J_L, avente priorità inferiore rispetto al primo, se e solo se J_L sta eseguendo delle istruzioni della sua sezione critica $z_{L,i} \in \beta^*_{H,L}$ quando J_H viene rilasciato.

Dimostrazione: *per ipotesi J_L sta eseguendo una sezione critica $z_{L,i}$ appartenente a $\beta^*_{H,L}$, che per definizione è l'insieme delle sezioni critiche di J_L che bloccano J_H. Pertanto l'esecuzione di J_H sarà arrestata: o a causa di un blocco diretto oppure perché, grazie al meccanismo dell'ereditarietà, J_L ha temporaneamente un livello di priorità maggiore rispetto a quello J_H.*

Lemma 1-2: la durata massima del blocco di J_H da parte di J_L, è definita da *una delle sezioni critiche appartenenti all'insieme* $\beta^*_{H,L}$, e non dipende dal numero di semafori che essi condividono.

Dimostrazione: *J_L blocca J_H, se e solo se il primo è in una sezione critica $z_{L,i} \in \beta^*_{H,L}$. Quando J_L esce da $z_{L,i} \in \beta^*_{H,L}$, allora J_H può esercitare la preemption su J_L, senza subire alcun nuovo blocco da parte dello stesso J_L.*

Teorema 1-3: si consideri il caso di un task set in cui J_0 è a priorità maggiore rispetto ai job $\{J_1,J_2,...,J_n\}$, i quali possono causare il blocco dello stesso J_0. Allora *il tempo di blocco B_{J0} di J_0* è definito attraverso *una delle sezioni critiche presenti in ciascun* $\beta^*_{0,i}$, per $i \in [1,n]$.

Dimostrazione*: considerando il Lemma 2-2, allora ogni J_i blocca J_0 al più per il tempo necessario ad eseguire una sezione critica $z_{0,i}$.*

Lemma 1-4: un semaforo S può bloccare in modalità *push-through* un job J, se e solo se ad S tentano di accedervi sia job con priorità inferiore a p(J), sia job con priorità superiore o uguale a quella di J.

Dimostrazione*: ciò è già stato dimostrato attraverso l'esempio #1 per l'applicazione di PI (figura 1.10).*

Tenendo ora presente che:

- $\beta^*_{i,j,k}$ rappresenta l'insieme delle *più lunghe sezioni critiche* eseguite da J_j ed associate ad S_k, che possono causare il *blocco diretto* o di tipo *push-through* di J_i;
- *$\beta^*i,.,k = \cup\beta^*_{i,j,k}$ con $P(J_j) < P(J_i)$* rappresenta l'insieme delle *più lunghe sezioni critiche bloccanti per J_i associate al semaforo S_k;*

Lemma 1-5: il job J_i può subire *un blocco descritto da un solo elemento di $\beta^*i,.,k$* di S_k , per ogni $k \in [1,m]$.

Dimostrazione: *supponendo che J_L sia in esecuzione della sezione critica associata all'utilizzo di S_K, e che J_H, con $p(J_H)>p(J_L)$, una volta lanciato, subisca conseguentemente un blocco da parte di J_L. Allora: non appena J_L rilascia la risorsa (e quindi il relativo semaforo S_K), J_H prenderà immediatamente possesso di S_K e potrà eseguire la propria sezione critica.*

In conclusione quindi: J_H potrà essere bloccato soltanto a causa di una delle sezioni critiche $\beta_{i,.,k}$ associate a S_K.*

Teorema 1-6: un job J che *richiede l'utilizzo di m distinti semafori* per la sua esecuzione, potrà subire al più *m blocchi* (si ricordi infatti il Lemma 1-5).

In virtù di quanto detto in precedenza, è possibile quindi stabilire un limite superiore al *massimo tempo blocco* che un job può subire. Ad esempio: se si considera un task set del tipo: $\{J_1,J_2,J_3,J_4\}$ (elencanti secondo *priorità decrescente*), nonché si suppone che i semafori condivisi siano $\{S_1,S_2,S_3\}$; allora: qualora i tre semafori di cui prima fossero già in possesso dei tre job a priorità inferiore, è facile intuire come *J_1 subirà nel peggiore dei casi un blocco dato complessivamente dalle tre sezioni critiche più lunghe* di J_2, J_3 e J_4.

1.14 Calcolo di un limite superiore per il tempo di blocco B_{Pi} di un task real-time P_i

L'*importanza della determinazione del parametro B_{Pi}* per ogni processo r-t è giustificata dal fatto che: solo attraverso *la loro precisa misura* è possibile realizzare *l'analisi di schedulabilità* per applicazioni real-time.

In letteratura esiste la possibilità di *calcolare un limite superiore al tempo di blocco* che può subire un processo r-t.

La metodologia che si intende ora illustrare è quella più *triviale,* nonché si può adoperare qualora *nessuno dei processi dell'applicazione r-t in esame abbia accessi annidati*.

Si definisce *tetto di priorità di una risorsa R_k* (noto anche come *priority ceiling*) PC_k:

$$PC_k = \max_j \{ p_j \mid P_j \text{ utilizza } R_k \}$$

- la *massima priorità tra quelle dei processi P_j che possiedono una sezione critica con utilizzo di R_k.* Tale priorità è conseguentemente quella *massima ereditabile* da un job qualora esso causi un blocco per questa risorsa.

Il *numero massimo di blocchi n_i* in cui un job J_i con *priorità p_i* può incorrere è:

$$n_i = max(l_i ; r_i)$$

con:

- $l_i =_{def}$ *"numero di job J_j con priorità $P(J_j) < p_i$ che accedono almeno ad una risorsa R_k con $PC_k \geq p_i$"*;
- $r_i =_{def}$ *"numero di risorse R_k con $PC_k \geq p_i$ accedute da almeno un job J_j con $P(J_j) < p_i$".*

A questo punto, *il calcolo dell'upper bound per il tempo di blocco B_{Pi}* avviene considerando *il valore minimo* assunto da una delle seguenti espressioni:

- $espressione_1 = \sum_{\forall j \mid p_j < p_i} \max_k \{ z_{j,k} \mid PC_k \geq p_i \}$;

- $espressione_2 = \sum_{\forall k \mid PC_k \geq p_i} \max_j \{ z_{j,k} \mid p_j < p_i \}$;
- $B_{Pi} = min \{ espressione_1 , espressione_2 \}$.

Nel seguito è mostrato un esempio in cui è applicato questo *metodo di stima*, nell'ipotesi che l'applicazione r-t di riferimento sia costituita da *processi periodici e schedulati secondo la politica RMPO*, nonché *nessun processo possieda accessi annidati* alle risorse condivise.

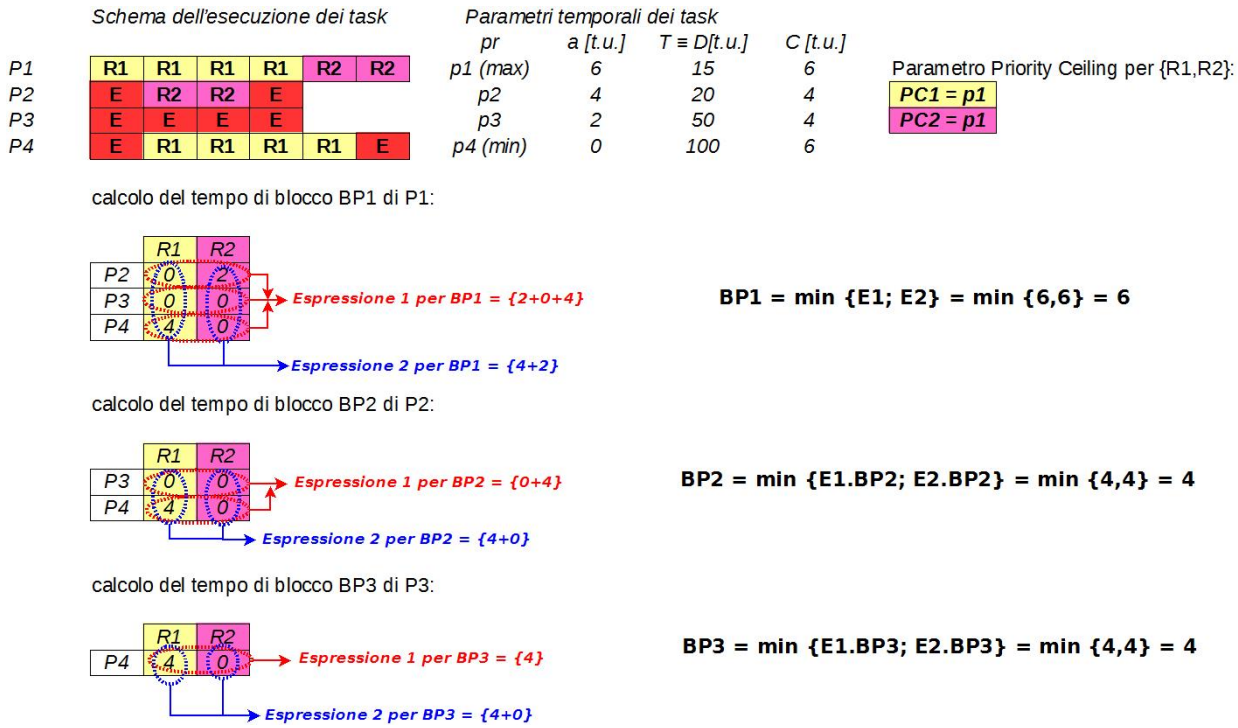


Figura 1.14 esempio di utilizzo pratico del metodo di stima dei B_{Pi} in esame

In realtà gli *upper bound* che si ottengono adoperando questa strategia di calcolo sono *laschi,* nonché spesso *imprecisi.* Questa affermazione è giustificata dalla semplice constatazione che:

- nell'espressione$_1$, è evidente che possano essere *inclusi dei contributi di processi differenti* relativi però *ad una stessa risorsa* R_k;
- nell'espressione$_2$, è facile osservare come invece i contributi bloccanti per P_i *di risorse diverse* possano essere in realtà *associate ad uno stesso processo* P_j.

La *complessità computazione* di questo *algoritmo per il calcolo del generico* B_{Pi} è $O(r_i l_i^2)$.

1.15 Calcolo del tempo di blocco con il metodo di Rajkumar

Un metodo che permette *la determinazione esatta* dei tempi di blocco $\{B_{P1}, B_{P2}, \ldots, B_{Pi}, \ldots\}$ di processi r-t i cui accessi alle risorse possono essere *annidati e non,* nonché regolati dal protocollo PI, è quello proposto da Rajkumar.

L'inconveniente principale di questo algoritmo di calcolo è rappresentato dalla sua *complessità computazionale*, di tipo *esponenziale*.

L'obiettivo di questo paragrafo è quello di descrivere i *passi necessari per la costruzione dell'albero di Rajkumar*, utile per la determinazione del tempo di blocco del generico processo P_i.

Questo strategia di calcolo si articola attraverso i seguenti passi:

- passo 1: dato P_i, si individuano *i processi* P_l *con* $p(P_j) < p(P_i)$ che *condividono risorse con lo stesso* P_i o con *task a priorità maggiore o uguale a* $p(P_i)$. A questo punto, si definisce un *simpleton*, cioè un *albero ad un livello*, per ogni coppia $\{P_i; P_l\}$ che riporti su proprio ramo il semaforo S_k (associato alla gestione della risorsa R_x condivisa da questi due processi), sia la durata della sezione critica bloccante z_{lk} di P_l per P_i;
- passo 2: indicato con P_k *il task a maggior priorità tra quelli inclusi nel set* {"P_l *che descrivono un simpleton per* P_i*"*} (vedi passo1), si utilizza allora questo processo come *nodo radice* per disporre, *secondo priorità decrescente*, i *restanti simpleton* $\{P_i;P_l\}$ (con $P_l \neq P_k$).

Eseguendo i due step di sopra si ottiene *l'albero di Rajkumar per* P_i.

A questo punto, è possibile determinare il *massimo tempo di blocco* B_{Pi} come la *somma dei pesi associati ai rami dell'albero*, tenendo però conto che:

- *caso 1:* se, nell'esame l'i-esimo ramo associato ad R_k, questa risorsa non è stata mai incontrata nel *percorso corrente di esplorazione dell'albero*, allora *il* $peso_{ramo\ i}$ *è sommato* a quelli che già definiscono B_{Pi}, e derivanti dall'analisi dei rami che precedono quello i-esimo;
- *caso 2:* se invece R_k del $ramo_i$ è già comparsa in un $ramo_j$ precedente a quello i-esimo, allora nella somma che definisce il parametro B_{Pi} è incluso il valore: *max* {*peso del* $ramo_i$; *peso del* $ramo_j$}.

L'algoritmo proposto da Rajkumar prevede pertanto l'esplorazione di *ogni percorso [nodo radice* $P_i \rightarrow$ *nodo foglia* P_k *dell'Albero di* P_i *]* e la determinazione del *corrispondente parametro* $B_{Pi,k}$. Infine, sarà considerato come *upper-bound del tempo di blocco* BP_i *di* P_i quello ottenuto come: $\max\{B_{Pi,1}; \ldots, B_{Pi,k}, \ldots\}$.

Per una migliore comprensione di quanto appena descritto si propone ora un semplice esempio. Più precisamente nella figura 1.15 seguente sono elencati i processi r-t che costituiscono l'applicazione in esame, le relative priorità e la durata degli accessi alle risorse $\{R_1, R_2\}$ che costituiscono il Resource Set.

	priorità	R1	R2
P1	p1 (max)	6	7
P2	p2	4	5
P3	p3	2	3
P4	p4 (min)		1

Figura 1.15 processi r-t, priorità e durata delle sezioni critiche per $\{R_1, R_2\}$

L'obiettivo è *la determinazione del massimo tempo di blocco* B_{P1} *del task* P_1. Ricordando il *passo 1* dell'algoritmo, è necessario determinare per P_1 i relativi *simpleton.* Essi sono rappresentati graficamente nella Figura 1.16.

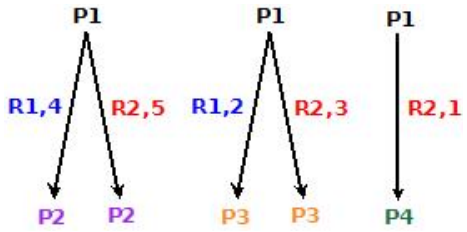


Figura 1.16 simpleton per il processo P_1

L'*Albero di Rajkumar di* P_1 si può ora facilmente ottenere dai *simpleton precedenti* ed esplorarlo per calcolare il B_{P1}.

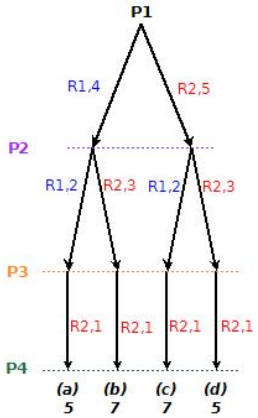


Figura 1.17 Struttura ad Albero di Rajkumar per il calcolo di B_{P1}

In conclusione, sommando *"opportunamente"* i pesi dei rami dell'Albero di P_1 (vedi le precisazioni fatte ai punti *caso 1* e *2* precedenti), è facile stabilire come il *massimo tempo di blocco* B_{P1} *sia pari a 7 t.u.* (vedi *percorsi b* e *c*).

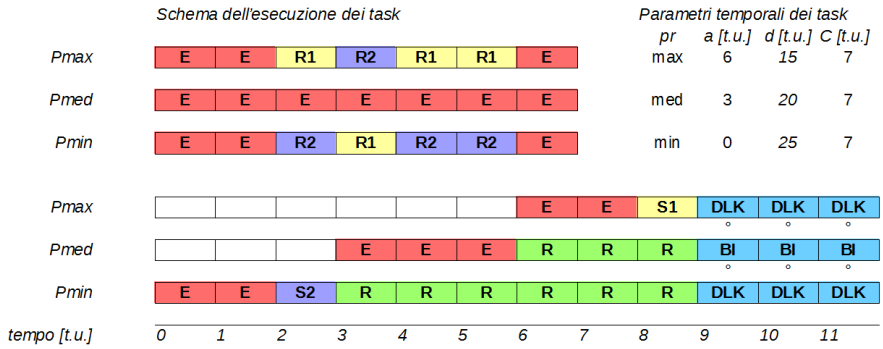
1.16 Problemi irrisolti dal protocollo PI

L'introduzione del protocollo PI per la gestione degli accessi agli elementi condivisi del Resource Set da parte di task r-t, lascia irrisolte *due questione*, e cioè:

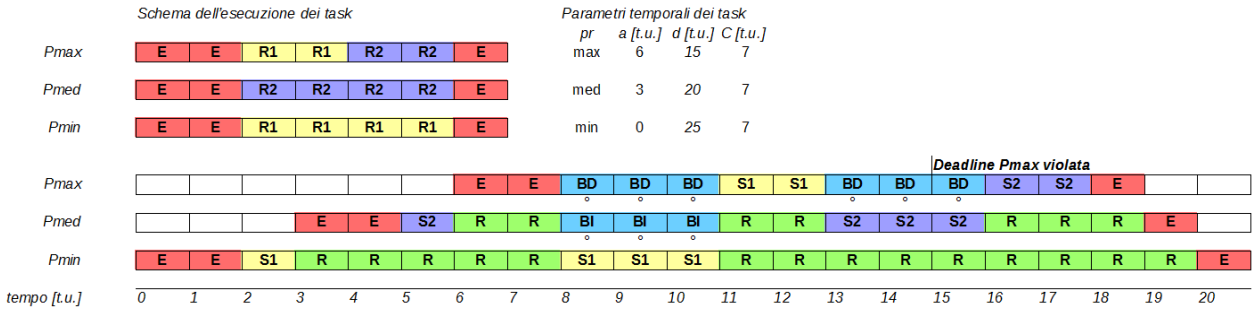
- l'occorrenza dei fenomeni di *deadlock;*
- la possibilità che si verifichino *concatenazioni di blocchi*.

Di seguito sono mostrati due applicazioni r-t ed i relativi diagrammi temporali che illustrano ciò graficamente.

(nota: con *BI* $=_{def}$ *"blocco indiretto"; DLK* $=_{def}$ *"deadlock"; °* $=_{def}$ *meccanismo di ereditarietà della priorità* $p(P_1)$)



(a)



(b)

Figura 1.18 Problemi irrisolti dal protocollo priority inheritance

Esiste in realtà *una soluzione al primo* di questi due problemi, e consiste nell'utilizzo di specifiche *tecniche per l'ordinamento totale degli accessi ai semafori delle risorse condivise.*

In sostituzione di PI è possibile adottare il protocollo *Priority Ceiling*, il quale non presenta nessuno di questi due fenomeni dannosi. Tuttavia *esso richiede la conoscenza a priori delle risorse richieste da ciascun processo r-t*. Questo requisito non sempre può essere soddisfatto per i sistemi real-time, e ciò rende il protocollo PI ancora largamente diffuso in questo ambito.

1.17 Modelli di Programmazione Lineare Intera per il calcolo dei tempi di blocco

Sinora si è descritta l'importanza della determinazione dei tempi di blocco per i task di un'applicazione r-t in cui è previsto l'utilizzo del protocollo Priority Inheritance, nonché *alcuni metodi di calcolo di questo parametro,* che tuttavia presentano inconvenienti non di poco conto.
In particolare, se si adottasse il *metodo esatto proposto da Rajkumar,* allora la complessità computazione associata all'utilizzo di questo algoritmo è esponenziale e ciò potrebbe rappresentare un insormontabile problema per le applicazioni r-t più complesse.

Le osservazioni appena fatte e la constatazione che il *calcolo del tempo di blocco* per un generico task P_i avviene: individuando le sezioni critiche z_{jk} dei processi P_j che possono potenzialmente bloccarlo, quindi selezionando quelle che *massimizzano il corrispondente* B_{Pi}; rendono possibile in questo ambito l'utilizzo del supporto della Ricerca Operativa, ed in particolare della Programmazione Lineare.
La Programmazione Lineare (PL) consente infatti l'identificazione di soluzioni ottime per problemi decisionali che possano essere rappresentati attraverso *un modello matematico*, e cioè più specificatamente mediante *un insieme di vincoli ed una funzione di ottimizzazione lineari*.
Il *vantaggio principale* di questo approccio consiste nel fatto che la risoluzione di un problema di PL ha *complessità polinomiale*, quindi particolarmente bassa anche per le applicazioni r-t più complicate.

Tuttavia è necessario osservare che le durate delle sezioni critiche z_{jk} sono espresse *in termini di unità temporali indivisibili*, e dunque è lecito aspettarsi che la soluzione ottima del problema *sia intera*.
Ciò implica che il *problema della determinazione dei tempi di blocco per task r-t che utilizzano PI sia* in realtà di *Programmazione Lineare Intera,* e che il calcolo della soluzione migliore preveda un maggior grado di complessità.

Senza entrare ulteriormente nel dettaglio di questioni che riguardano la Ricerca Operativa ed i relativi metodi di calcolo, alla base delle scelte progettuali fatte vi è la constatazione che nei

modelli ILP è possibile introdurre ed utilizzare delle *variabili binarie*, cioè *forzate ad assumere solamente uno tra i valori interi {0;1}* per mezzo di vincoli lineari che descrivono *usuali relazioni logiche del calcolo proposizionale*.

In particolare, si consideri una *"generica proposizione elementare a"*, allora: è indicata con *x(a) la variabile logica* che *assume valore unitario se "a è vera"*; e *valore nullo quando "a è falsa"*.

A questo punto, le relazioni logiche di *negazione*, *implicazione*, *unione ed intersezione tra le generiche proposizioni {a, b, c, ... }*, cui come detto in precedenza *corrispondono le variabili proposizionali {x(a), x(b), x(c), ... }*, sono descritte dai seguenti set di disequazioni ed equazioni lineari:

- *Negazione*. La relazione logica "b = $\bar{a}$" è modellata mediante il vincolo: *"x(a) + x(b) = 1"*, *con: {x(a), x(b)} ∈{0,1}*.

  Dimostrazione: se "x(a) = 1" ⇒ "x(b) = 1 – 1 = 0", e se "x(a) = 0" ⇒ "x(b) = 1 – 0 = 1";
- *Implicazione*. La relazione logica "a ⇒ b" ("a implica b") è modellata attraverso la disuguaglianza: *"x(b) ≥ x(a)", con: {x(a), x(b)} ∈{0,1}*.

  Dimostrazione: se "x(a) = 1", allora: "x(b) ≥ 1".
- *Unione (Or Logico)*. Date due proposizioni a e b, la variabile "c = a ∨ b" è rappresentata con le relazioni lineari:
  - *"x(c) ≥ x(a)", "x(c) ≥ x(b)", "x(c) ≤ x(a) + x(b)", con x(c) ∈{0, 1}*.

  Dimostrazione: le prime due disequazioni vincolano x(c) ad assumere valore "1" se una tra x(a) e x(b) non è nulla. *La terza relazione forza invece x(c) a "0" qualora entrambe le variabili x(a) e x(b) siano nulle.*
- *Intersezione.* Date due proposizioni a e b, la variabile "c = a ∧ b" è rappresentata con le relazioni lineari:
  - *"x(c) ≤ x(a)", "x(c) ≤ x(b)", "x(c) ≥ x(a) + x(b) − 1", x(c) ∈{0, 1}*.

  Dimostrazione: Le prime due disequazioni impongono per x(c) il valore "0" nel caso che una delle due variabili {x(a), x(b)} valga "0". La terza relazione impone invece "x(c) = 1" se "x(a) = x(b) = 1".

Quanto illustrato sinora è di fondamentale importanza poiché consente la rappresentazione nei modelli ILP, con corrispondenti variabili binarie, di opportune variabili proposizionali che indicano *"la presenza o meno di specifiche proprietà per processi r-t"*.

Inoltre, sempre grazie al supporto dato dall'introduzione di variabili binarie, è possibile modellare opportunamente *le caratteristiche delle interazioni bloccanti tra essi.*

Nei capitoli successivi si discuterà dei ragionamenti alla base dell'insieme di regole ideate con cui è stato possibile definire modelli lineari di Programmazione Lineare Intera utili alla stima dei tempi di blocco B_{Pi} dei task P_i di una generica applicazione r-t, che preveda la condivisione di risorse tra processi con priorità differenti e l'adozione del protocollo Priority Inheritance.
Per far ciò, si è reso utile articolare il lavoro come descritto nei passaggi successivi:

1. classificazione delle proprietà principali e modellazione dei comportamenti dei processi $\{P_i, P_j, P_k, ...\}$ di un generico task set;
2. ideazione di vincoli lineari che permettano di tener conto della presenza dei possibili blocchi tra i processi;
3. definizione di uno specifico insieme di vincoli per il processo P_a di cui si intende stimare il tempo di blocco B_{Pa};
4. determinazione di un sistema di penalizzazione per le *sequenze di blocchi* che si dimostrano essere *inammissibili* analizzando l'ordine degli accessi alle risorse da parte di P_a e dei processi a minor priorità, potenzialmente bloccanti per lo stesso P_a.

CAPITOLO II

Modellazione dei processi di un'applicazione real-time

2.1 Introduzione alla rappresentazione dei processi

Il primo problema che si incontra nella definizione di un modello ILP per la stima del massimo tempo di blocco B_{P1} del processo più critico P_1, è la *rappresentazione dei processi P_i inclusi nel task dell'applicazione r-t in esame, nonché potenzialmente bloccanti per P_1.*

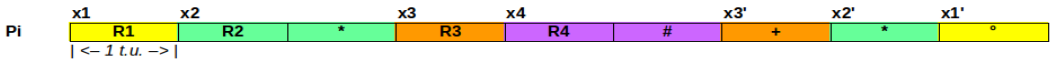
In questo ambito, per rendere facilmente comprensibili le scelte progettuali fatte, risulta conveniente prendere in considerazione i seguenti casi di studio:

- C.d.S. I: si suppone che l'insieme delle risorse di cui necessita il generico processo P_i sia $\{R_1,R_2,R_3,R_4\}$, e che ad esse vi accede secondo lo schema $[R_1{:}9\ [R_2{:}7\ [R_3{:}4\ [R_4{:}2]]]]]$;
- C.d.S. II: si ipotizza che l'insieme delle risorse di cui necessita il generico processo P_i sia $\{R_1,R_2\}$, e che ad esse vi accede come $[R_1{:}3\ [R_2{:}1]]$;
- C.d.S. III: si suppone che l'insieme delle risorse di cui necessita il generico processo P_i sia $\{R_1,R_2,R_3\}$, e che ad esse P_i vi accede secondo lo schema $[R_1{:}3\ [R_2{:}2\ [R_3{:}1]]]$.

La modellazione ILP del secondo ed il terzo caso di studio appena citati, oltre a rappresentare ulteriori fonti di chiarimento circa i ragionamenti fatti per la modellazione del task relativo al primo degli esempi di seguito analizzati, risultano in realtà particolarmente utili in quanto adoperati nelle analisi condotte nei capitoli successivi, dedicati ai problemi della modellazione delle interazioni bloccanti tra task.

Infine prima di proseguire con la discussione, si precisa che un'ulteriore particolare esempio di modellazione sarà analizzato al termine di questo capitolo, in quanto è possibile che nei Task Set di applicazioni r-t siano inclusi processi P_k con più sezioni critiche *"esterne"*.

2.2 Caratteristiche di P_i per il caso di studio I



notazione:

R_1, °={*esecuzione di istruzioni con lock del semaforo di R_1*};

R_2, *={*esecuzione di istruzioni con lock del semaforo di R_2*};

R_3, +={*esecuzione di istruzioni con lock del semaforo di R_3*};

R_4, #={*esecuzione di istruzioni con lock del semaforo di R_4*};

Figura 2.1 Schema della modalità di accesso a $\{R_1, R_2, R_3$ e $R_4\}$ da parte di P_i

Per il C.d.S I, si suppone che il processo P_i abbia le seguenti caratteristiche:

- il suo tempo di esecuzione complessivo è di 9 time unit (t.u.);
- è composto da quattro sezioni critiche x_1, x_2, x_3 e x_4 di durata rispettivamente pari a 9, 7, 4 e 2 t.u., che prevedono l'accesso annidato e l'utilizzo delle risorse R_1, R_2, R_3 e R_4, come mostrato in figura 2.1.

La prima scelta progettuale effettuata consiste nel *considerare singolarmente le sezioni critiche x_1, x_2, x_3 e x_4 di P_i,* e di immaginare che l'esecuzione del processo in esame possa essere rappresentata come una combinazione z_k di $\{x_1, x_2, x_3, x_4\}$, compatibile con un'analoga modellazione dei processi P_j che appartengono allo stesso task set di P_i, nonché potenzialmente detentori o richiedenti del lock delle risorse utilizzate da P_i.

2.3 Combinazioni di esecuzione z_k di P_i

	x1	x2	x3	x4	x3'	x2'	x1'
z1	1	0	0	0	0	0	0
z2	0	1	0	0	0	0	1
z3	0	0	1	0	0	1	1
z4	0	0	0	1	1	1	1
z5	0	0	0	0	1	1	1
z6	0	0	0	0	0	1	1
z7	0	0	0	0	0	0	1
z8	0	1	0	0	0	0	0
z9	0	0	1	0	0	1	0
z10	0	0	0	1	1	1	0
z11	0	0	0	0	1	1	0
z12	0	0	0	0	0	1	0
z13	0	0	1	0	0	0	0
z14	0	0	0	1	1	0	0
z15	0	0	0	0	1	0	0
z16	0	0	0	1	0	0	0

Figura 2.2 Tabella delle combinazioni di esecuzione z_k associate al processo P_i

La tabella mostrata di sopra elenca tutte le possibili z_k che possono essere utilizzate per la descrizione dell'esecuzione di P_i, nonché è possibile notare come:

a) la presenza di un *valore unitario* in corrispondenza dell'elemento (k,j) della tabella, implichi che la corrispondente combinazione *z_k preveda l'esecuzione della sezione critica j-esima x_j* per P_i;
b) la presenza di un *valore nullo* in corrispondenza dell'elemento (k,j) della tabella, implichi che la corrispondente combinazione *z_k non preveda l'esecuzione della sezione critica j--esima x_j* per P_i.

In realtà, è facile osservare come nella tabella 2.1 siano presenti anche colonne destinate a rappresentare la potenziale esecuzione delle *sezioni critiche pendenti x_j'*, il cui significato però sarà chiarito successivamente.

La regola alla base della definizione di ciascuna z_k di P_i è pertanto la seguente:

- (Regola 2.1) "*una combinazione z_k è associata all'esecuzione di una ed una sola delle sezioni critiche x_j di P_i (vedi: z_1, z_8, z_{13}, z_{16}); o alternativamente è costituita dall'esecuzione di una ed una sola x_j e di un insieme specifico di sezioni critiche pendenti indicate con x_j' (vedi: z_2, z_3, z_4, z_9, z_{10}, z_{14}); o ancora, essa può essere descritta solamente dall'esecuzione di un insieme specifico di sezioni critiche pendenti (vedi: z_5, z_6, z_7, z_{11}, z_{12}, z_{15})*".

Per comprendere la regola appena enunciata, si considerino i seguenti scenari d'esecuzione:

1. si supponga che il processo P_i abbia consumato solo una frazione trascurabile della prima t.u. del suo tempo di esecuzione complessivo, e tale da rendere quindi valida l'assunzione che P_i possieda il lock della risorsa R_1, ma non quello delle risorse R_2, R_3 e R_4.
   In quest'ipotesi, se esistesse nel task set un processo P_k con priorità $p(P_k)>p(P_i)$ che richieda la risorsa R_1, allora P_i sarebbe considerato il processo che causa il blocco diretto sulla risorsa R_1, e specificatamente per la *richiesta di R_1 inoltrata da P_k*.
   Inoltre, in questo caso si può assumere che *di P_i risulti eseguita solamente la sezione x_1*, dato che:
   - indipendentemente che P_i subisca o meno dei *blocchi diretti* per le risorse R_2, R_3 ed R_4, la *modalità annidata* con cui questo processo accede alle risorse di cui necessita (x_1 è la sezione più esterna), rende del tutto trasparente a P_k l'esecuzione di x_2, x_3 e x_4 da parte di P_i quando esso esegue x_1.
     Se P_i subisse infatti dei blocchi diretti sulle risorse R_2, R_3 e/o R_4, supposte in possesso di un altro processo P_h con $p(P_h)< p(P_k)$, allora tale blocco *diretto* subito da P_i, e *transitivo* per P_k, sarà modellato attraverso la presenza di un'ulteriore combinazione z_h associata all'esecuzione di ciascuna eventuale sezione critica x_h del processo P_h, bloccante per P_i e conseguentemente anche per P_k.

   Questo ragionamento è alla base della definizione della combinazione z_1 ($x_1=1$; $x_j=0$ con $j\neq1$; $x_k'=0$ $\forall k'$) e, attraverso facili deduzioni, anche delle combinazioni z_8, z_{13} e z_{16} ($x_i=1$; $x_j=0$ con $i \in \{2, 3, 4\}$ e $j\neq1$; $x_k'=0$ $\forall k'$). In questi ultimi tre casi P_i si supporrà in possesso rispettivamente anche di R_2, R_3 ed R_4, richieste da un task a più alta priorità come P_k.

2. si supponga che del processo P_i sia stata interamente eseguita la prima t.u. del suo execution time complessivo, quindi sia stata consumata solo una frazione trascurabile

della sua sezione critica x_2. In questo caso, P_i possiede il lock dei semafori che gestiscono gli accessi alle risorse R_1 e R_2, ma non quelli delle risorse R_3 e R_4.

Esattamente come nello scenario al *punto1*, si suppone che un processo P_k richieda l'utilizzo della risorsa R_1, quindi subisca conseguentemente il blocco diretto determinato da P_i.

In questo caso, l'esecuzione della sola x_2 da parte di P_i non sarebbe sufficiente a modellare il blocco complessivamente subito da P_k per causa di P_i, perché:

- essendo richiesta da P_k la risorsa R_1, se si supponesse eseguita solamente x_2, allora P_i conserverebbe il possesso di R_1, e libererebbe solamente la risorsa R_2;
- x_2, essendo annidata in x_1, ha estensione temporale inferiore a quella di x_1, quindi non si considererebbe il complessivo *contributo bloccante* definito da P_i;
- si rende necessario in questo caso che di P_i sia eseguita interamente x_2, ed assieme ad essa anche la *sezione critica pendente* x_1' (vedi figura 2.1), rappresentante per l'appunto le t.u. di P_i da eseguire dopo il rilascio di R_2 necessarie affinché questo stesso processo liberi anche la risorsa R_1.

Lo scenario appena analizzato è rappresentato dalla combinazione z_2 ($x_2=1$ e $x_j=0$ $\forall j\neq 2$, $x_1'=1$ e $x_j'=0$ $\forall j'\neq 1$). Inoltre, con ragionamenti analoghi, è possibile determinare anche le combinazioni:

- z_3, per la quale si suppone che sia stato eseguito P_i sino a che esso abbia occupato la risorsa R_3, nonché che il processo P_k richieda la risorsa R_1, implicando così che il tempo di blocco definito da P_i sia descritto dalla somma delle durate della sezione critica x_3 e di quelle pendenti x_2' e x_1', fondamentali affinché sia liberata la risorsa R_1;
- z_4, per la quale si suppone che sia stato eseguito P_i sino a che esso abbia occupato la risorsa R_4, nonché che il processo P_k richieda la risorsa R_1, implicando così che il tempo di blocco definito da P_i sia descritto dalla somma delle durate della sezione critica x_4 e di quelle pendenti x_3', x_2' e x_1', fondamentali affinché sia liberata la risorsa R_1;
- z_9, per la quale si suppone che sia stato eseguito P_i sino a che esso abbia occupato la risorsa R_3, nonché che il processo P_k richieda la risorsa R_2, implicando così che il tempo di blocco definito da P_i sia descritto dalla somma delle durate della sezione critica x_3 e di quella pendente x_2', fondamentali affinché sia liberata la risorsa R_2;

- z_{10}, per la quale si suppone che sia stato eseguito P_i sino a che esso abbia occupato la risorsa R_4, nonché che il processo P_k richieda la risorsa R_2, implicando così che il tempo di blocco definito da P_i sia descritto dalla somma delle durate della sezione critica x_4 e di quelle pendenti x_3' e x_2', fondamentali affinché sia liberata la risorsa R_2;
- z_{14}, per la quale si suppone che sia stato eseguito P_i sino a che esso abbia occupato la risorsa R_4, nonché che il processo P_k richieda la risorsa R_3, implicando così che il tempo di blocco definito da P_i sia descritto dalla somma delle durate della sezione critica x_4 e di quella pendente x_3', fondamentale affinché sia liberata la risorsa R_3.

2.4 Analisi del ruolo delle sezioni critiche pendenti x_j'

Dall'esame degli esempi illustrati nel paragrafo precedente si comprende come ciascuna *sezione critica pendente* x_j' sia in realtà un *residuo* di una corrispondente sezione x_j del processo P_i, avente specificatamente le seguenti caratteristiche:

1. x_j, associata all'utilizzo di una generica risorsa R_i da parte di P_i, prevede l'accesso annidato ad una o più risorse $\{R_k, R_l, R_m, \ldots\}$, secondo lo schema $[(x_j,R_j):\delta\ x_j\ [(x_{j+1},R_k):\delta\ x_{j+1}\ [(x_{j+2};R_l):\delta\ x_{j+2}\ [\ldots]\ ]\ ]\ ]$;
2. il rilascio della risorsa R_j da parte di P_i, e la conseguente terminazione della sezione x_j,avviene solo dopo che lo stesso P_i abbia effettuato quello di ciascuna delle risorse R_k accedute in modalità annidata. Quindi, l'estensione temporale della sezione x_j si assume sicuramente maggiore di quella di ogni $x_{j+1, j+2, \ldots}$ che essa include internamente.

L'utilizzo delle *sezioni critiche pendenti* x_j' è quindi fondamentale per ogni processo P_i appartenente al task set in esame, che preveda la possibilità che esso, nel corso della propria esecuzione, possa rilasciare una o più risorse, ma rimanere in possesso ugualmente del lock di altre.

Per comprendere meglio l'utilità delle sezioni critiche pendenti x_j', si consideri la combinazione z_5:

- essa modella lo scenario per cui P_i abbia già eseguito la sezione x_4, rilasciando il lock della risorsa R_4, ma sia altresì ancora in possesso delle risorse R_1, R_2 e R_3. Ergo, se ci fosse un processo P_k con $p(P_k)>p(P_i)$, che richiedesse il lock della risorsa R_1, allora di P_i *sarebbe necessario eseguire le sezioni critiche pendenti* x_3', x_2' *e* x_1' (esattamente secondo questo ordine), poiché:

a) solamente in corrispondenza della fine di x_3' avviene il rilascio della risorsa R_3 (anche se non richiesta da P_k);
b) solamente in corrispondenza della terminazione di x_2' avviene il rilascio della risorsa R_2 (anche se non è di interesse per P_k);
c) in ultimo, solo quando anche la sezione x_1' risulterà eseguita (quindi sarà conseguentemente terminato il processo P_i), allora ci sarà il rilascio di R_1, il cui lock sarà pertanto disponibile per P_k.

Tale ragionamento giustifica quindi l'introduzione della combinazione z_5 nel modello ilp qualora sia presente nel task set un processo come P_i.

Inoltre, non bisogna tralasciare il fatto che l'*esecuzione in sequenza di x_3', x_2' e x_1'*, sia in realtà una diretta conseguenza della struttura con annidamento di questo stesso task. Infatti, se per P_i ci fosse una sezione critica x_j che prevedesse l'utilizzo della risorsa R_j, quindi l'accesso in modalità annidata a R_k come $[(x_j,R_j)$: δ x_j $[(x_k,R_k)$: δ $x_k]]$, ma questo processo rilasciasse R_j e R_k *contemporaneamente*, allora per x_j non sarebbe necessario definire alcuna *sezione critica pendente x_j'* nel modello, dato che non potrebbe mai verificarsi la situazione per cui P_i producendo un blocco per R_k la rilascia, ma rimane in possesso del lock di R_j (nota: si assume che x_j inizi prima rispetto alla sezione x_k, ma termini nel medesimo istante).

Con ragionamenti analoghi a quello appena fatto, è possibile dedurre l'importanza dell'introduzione nel modello ilp, per quanto concerne il processo P_i, anche delle combinazioni z_k:

- z_6, modellante lo scenario per cui P_i abbia già utilizzato e rilasciato le risorse $\{R_3,R_4\}$, ma possiede il lock di $\{R_1,R_2\}$, di cui la prima è richiesta da un processo P_k con $p(P_k) > p(P_i)$;
- z_7, modellante lo scenario per cui P_i abbia già utilizzato e rilasciato le risorse $\{R_2,R_3,R_4\}$, ma possiede il lock di R_1, richiesta da un processo P_k con $p(P_k) > p(P_i)$;
- z_{11}, modellante lo scenario per cui P_i abbia già utilizzato e rilasciato la risorsa R_4, ma possiede il lock di $\{R_1,R_2,R_3\}$, ed R_2 sia richiesta da un processo P_k con $p(P_k) > p(P_i)$;
- z_{12}, modellante lo scenario per cui P_i abbia già utilizzato e rilasciato le risorse $\{R_3,R_4\}$, ma sia in possesso del lock di $\{R_1,R_2\}$, ed R_2 sia richiesta da un processo P_k con $p(P_k)>p(P_i)$;
- z_{15}, modellante lo scenario per cui P_i abbia già utilizzato e rilasciato la risorsa R_4, ma sia in possesso del lock di $\{R_1,R_2,R_3\}$, ed R_3 sia richiesta da un processo P_k con $p(P_k) > p(P_i)$;

2.5 Catene di attivazione del processo P_i

L'introduzione delle *sezioni critiche pendenti* x_j' per P_i è ben giustificata anche dal fatto che con esse è possibile modellare efficientemente le *catene di attivazione di processo*.

Per cercare di spiegare in modo semplice che cosa si intenda per *catena di attivazione di processo*, si consideri la figura 2.3 di sotto, associata allo scenario di esecuzione di P_i tale per cui:

1. il processo P_i sia stato già eseguito sino a che non risulti consumata una frazione trascurabile della sua sezione critica x_4, così da possedere il lock delle risorse $\{R_1,R_2,R_3,R_4\}$;
2. l'insieme degli istanti con cui sono scanditi gli eventi è quello {t(a), t(b), ..., t(h)}, e per essi risultano verificate le relazioni: {“t(a) precede t(b)”; “t(b) precede t(c)”; “t(c) precede t(d)”; “t(d) precede t(e)”; “t(e) precede t(f)”; “t(f) precede t(g)”; “t(g) precede t(h)”};
3. si suppone che esista un processo P_k, con $p(P_k) > p(P_i)$, che all'istante t(a) richieda il possesso della risorsa R_4, implicando così che P_i esegua *solamente la sezione critica* x_4 e rilasci conseguentemente R_4, ma continui altresì a detenere il lock del set di risorse $\{R_1,R_2,R_3\}$ in corrispondenza dell'istante t(b);
4. successivamente, si ipotizza che P_k richieda in corrispondenza degli istanti t(c), t(e) e t(g) le risorse R_3, R_2 ed R_1, e che conseguentemente di P_i siano eseguite in modo *“discontinuo”* le sezioni critiche pendenti x_3', x_2', x_1', consentendo così il rilascio del lock di R_3, R_2 e R_1 rispettivamente negli istanti t(d), t(f), t(h).

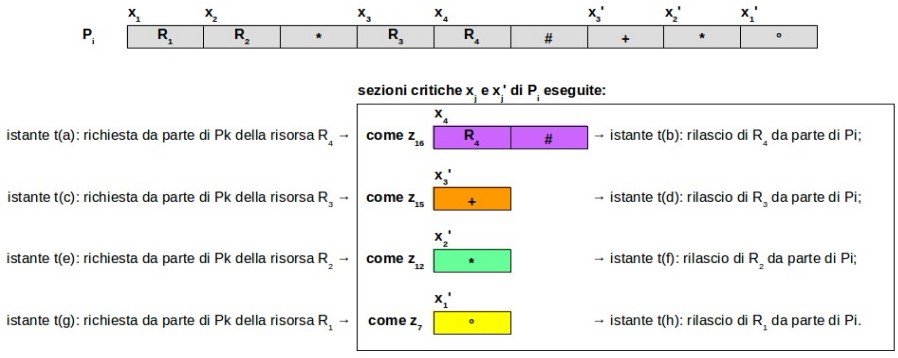


Figura 2.3 Esempio di catena di attivazione per il processo P_i

A questo punto, il Lettore potrebbe considerare valida l'idea che la medesima esecuzione delle sezioni critiche $\{x_4 \rightarrow x_3' \rightarrow x_2' \rightarrow x_1'\}$ di P_i, ottenuta attraverso l’utilizzo della sequenza ordinata $\{z_{16} \rightarrow z_{15} \rightarrow z_{12} \rightarrow z_7\}$ appena discussa, possa essere identicamente definita mediante l'utilizzo

della sequenza di z_k: $\{z_{16} \rightarrow z_5\}$.

In realtà, questa considerazione è errata dato che, ricordando il ragionamento relativo alla descrizione del ruolo svolto da z_5 nella modellazione di P_i, allora la *catena* $\{z_{16} \rightarrow z_5\}$ si può *attivare* solo se si verificano le seguenti condizioni:

1. il processo P_i possiede il controllo delle risorse $\{R_1,R_2,R_3,R_4\}$;
2. il processo P_k richiede all'istante t(a) la risorsa R_4, cosicché P_i può difatti eseguire la sezione critica x_4 utilizzando la rappresentazione fornita dalla combinazione z_{16};
3. il processo P_k richiede all'istante t(c) non più R_3, bensì la risorsa R_1, consentendo così a P_i di eseguire tutte le sezioni critiche che portino al rilascio di R_1. In questo caso quindi è evidente che non vi è un'esecuzione *discontinua* delle sezioni critiche pendenti x_3', x_2' e x_1', e che l'esecuzione delle prime due sezioni critiche pendenti ed il conseguente rilascio di R_3 e R_2 sia determinata dalla richiesta di una risorsa non corrispondente né ad R_2 , né ad R_3.

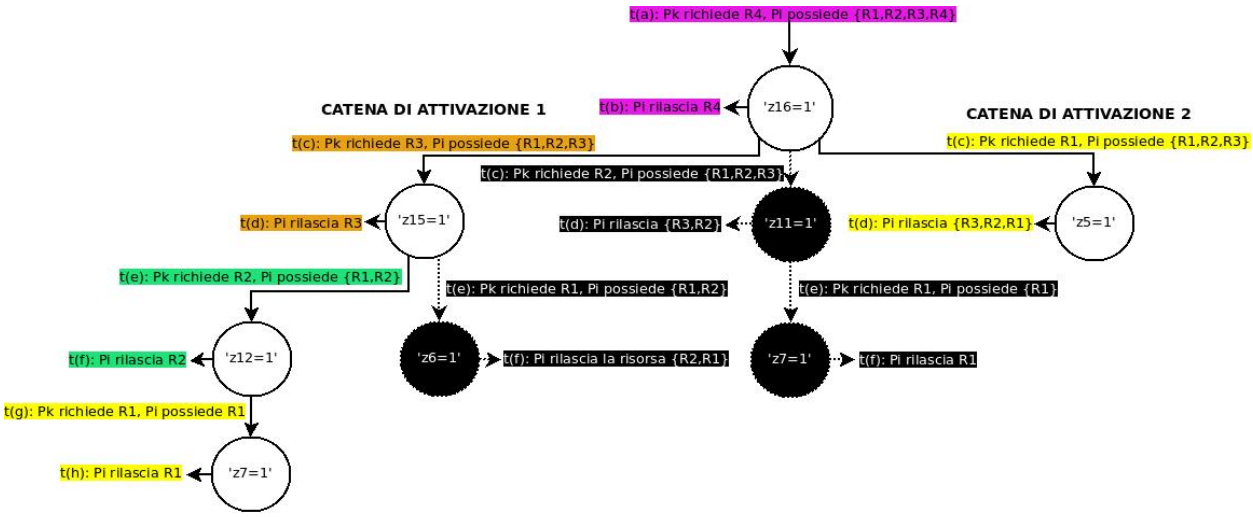


Figura 2.4 Rappresentazione delle catene di attivazione dell'esempio precedente

Nella diagramma ad albero in figura 2.4, oltre ad essere rappresentate graficamente le *catene di attivazione* discusse nell'esempio precedente, sono evidenziati dei *nodi in neretto e connessi all'albero attraverso degli archi tratteggiati*. Essi sono inclusi nel grafico in questione perché in realtà concorrono alla formazione di altre *catene di attivazione* per quanto riguarda il processo P_i.

In particolare, la struttura completa delle *catene di attivazione di* P_i è illustrata nella figura successiva.

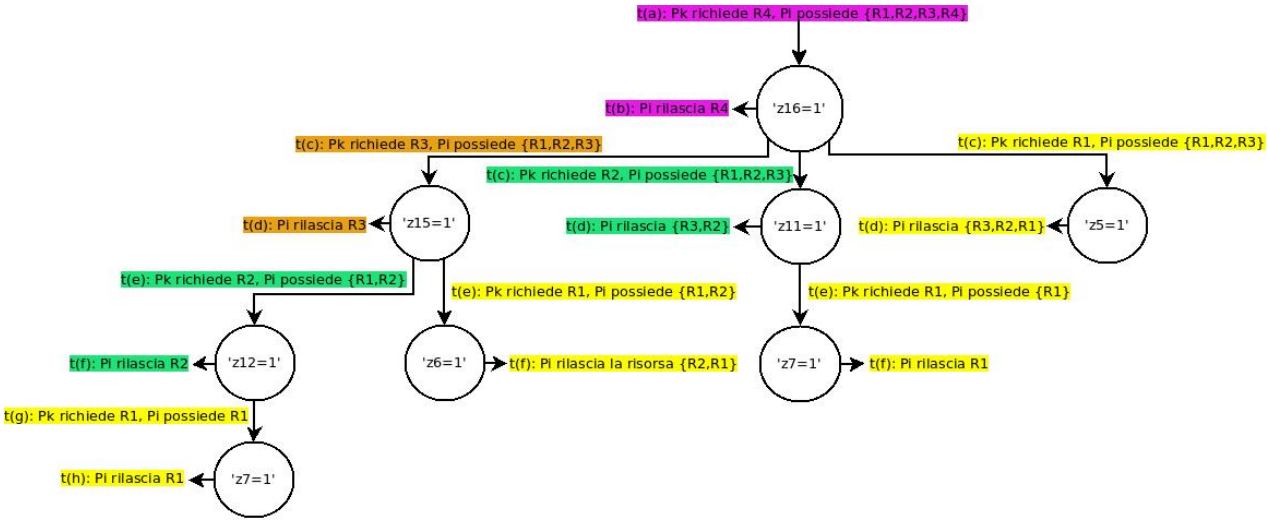


Figura 2.5 Albero delle catene di attivazione del processo P_i

L'interpretazione di questi diagrammi è semplice e si basa sulle seguenti assunzioni:

- dato un nodo qualsiasi della struttura in esame, ogni *arco incidente in esso* indica quale sia il set di risorse posseduto da P_i e quale risorsa sia stata richiesta da un processo P_k a priorità più elevata, affinché la combinazione di esecuzione z_k indicata al suo interno possa essere considerata eseguita per il processo P_i;
- dato un generico nodo della struttura in esame, ogni *arco uscente da esso* indica quale risorsa sia liberata da P_i, dopo appunto l'esecuzione delle sezioni critiche x_j ed eventualmente x_j' associate alla combinazione z_k di nodo.

In realtà, per non rendere la figura 2.5 eccessivamente complessa, in essa non sono mostrati due ulteriori contesti esecutivi che possono verificarsi qualora P_i sia incluso tra i processi del Task Set di una possibile applicazione r-t. E cioè, se P_i fosse presente tra quelli bloccanti per P_k, allora potrebbero verificarsi i seguenti scenari:

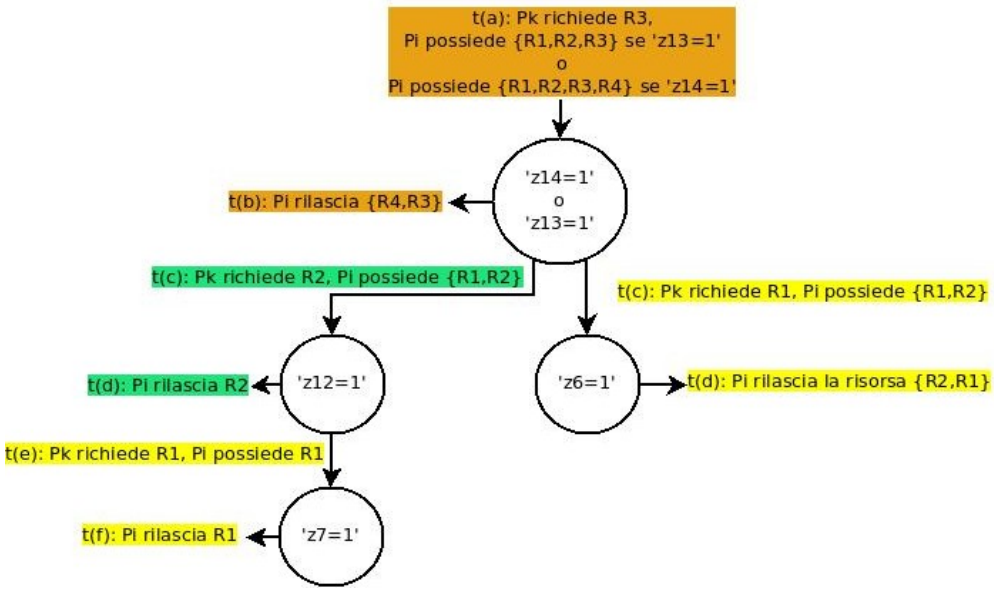


Figura 2.6 Albero delle catene di attivazione di P_i nello scenario I

1. *scenario I*: si supponga che il processo P_i sia in possesso di $\{R_1,R_2,R_3\}$, e potenzialmente anche della risorsa R_4. In questo caso, se uno o più processi $\{P_k, P_j, \ldots\}$ richiedessero in sequenza:
   a) le risorse $\{R_3 \rightarrow R_2 \rightarrow R_1\}$, allora sarebbe *"attivata la catena"* di combinazioni $\{(z13/z14) \rightarrow z_{12} \rightarrow z_7\}$;
   b) le risorse $\{R_3 \rightarrow R_1\}$, allora sarebbe *"attivata la catena"* di combinazioni $\{(z_{13}/z_{14}) \rightarrow \rightarrow z_6\}$.

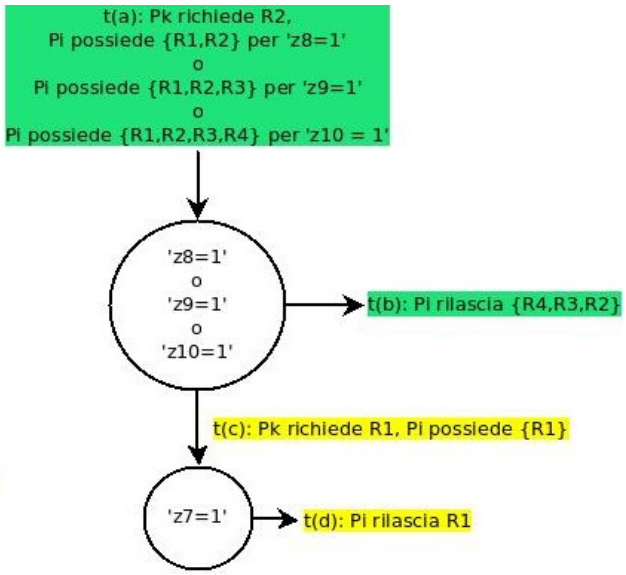


Figura 2.7 Albero delle catene di attivazione di P_i nello scenario II

2. *scenario II*: si supponga che il processo P_i sia in possesso di $\{R_1,R_2\}$, e potenzialmente

anche delle risorse R_3 ed R_4. In questo caso, se uno o più processi $\{P_k, P_j, \ldots\}$ richiedessero le risorse $\{R_2 \rightarrow R_1\}$ in questa sequenza, allora sarebbe *"attivata la catena"* di combinazioni $\{(z_8/z_9/z_{10}) \rightarrow z_7\}$.

2.6 Sintesi delle informazioni relative alle catene di attivazione del processo P_i

Le informazioni presenti nelle strutture ad albero per la gestione delle sezioni critiche pendenti di un processo, possono essere rappresentate sinteticamente in una tabella. In particolare nella figura 2.8 di sotto è mostrata quella ottenuta sulla base dei diagrammi ad albero associati al processo di riferimento P_i.

	R1	R2	R3	R4
z1	*	*	*	*
z2	*	*	*	*
z3	*	*	*	*
z4	*	*	*	*
z5	*	*	*	*
z6	*	*	*	*
z7	*	*	*	*
z8	x1', z7	*	*	*
z9	x1', z7	*	*	*
z10	x1', z7	*	*	*
z11	x1', z7	*	*	*
z12	x1', z7	*	*	*
z13	x1', z6	x2', z12	*	*
z14	x1', z6	x2', z12	*	*
z15	x1', z6	x2', z12	*	*
z16	x1', z5	x2', z11	x3', z15	*

Figura 2.8 Tabella riassuntiva delle informazioni relative alle catene di attivazione di P_i

La sua interpretazione si basa sulla semplice regola per cui: ciascuna cella non vuota associata alla *riga* z_k ed alla *colonna* R_x indica *quale sia la combinazione* z_j *di* P_i *che sarà eseguita qualora fosse richiesto il lock della risorsa* R_x e supposto che il processo P_i sia stato già in parte eseguito secondo le modalità descritte dalla z_k di riga.

Le *celle vuote* (segnate con *"*"*) implicano invece che in corrispondenza dell'esecuzione di P_i come descritto dalla z_k di riga, il task in questione non potrà essere considerato come *rispondente* ad una richiesta di liberazione della risorsa R_x *di colonna,* e quindi per esso *non sarà prevista l'esecuzione di alcuna sua sezione critica.*

In realtà, la fondamentale importanza delle sezioni critiche pendenti x_j' non è evidente qualora si prenda in considerazione processi che abbiano schemi di accesso alle risorse condivise come quello ipotizzato per il caso di studio I in esame.

In queste situazioni infatti, l'introduzione delle variabili binarie e dei vincoli lineari che permettono di rappresentare il comportamento di P_i qualora rimanga in possesso di risorse relative a sezioni critiche *esterne*, non è determinante per la corretta stima del tempo di blocco B_{P1}.

A dimostrazione di quanto appena affermato si consideri il seguente scenario:

1. se il processo P_i del caso di studio I provocasse un blocco diretto per la risorsa R_2 attraverso la presenza della sua combinazione z_8 con valore unitario in soluzione ottima finale, allora P_i in realtà non potrebbe causare un blocco anche per il possesso del lock della risorsa R_1 attraverso la z_7.

   Infatti è semplice osservare come in termini di massimizzazione del tempo di blocco B_{P1} di un generico processo P_1 con $p(P_1) > p(P_i)$:

   a) tra un singolo blocco diretto su R_1 con peso 9 t.u. (vedi durata temporale δx_1);
   b) e un doppio blocco diretto $\{R_2 \rightarrow R_1\}$ come $\{z_8 = 1; z_7 = 1\}$ con peso complessivo 8 t.u. (vedi estensione delle sezioni x_2 e x_1');

   ovviamente *sarà sempre favorita* la scelta del blocco al punto a) rispetto a quello b), nonché P_i potrà risultare bloccante *per la risorsa R_2* attraverso la z_8, se e solo se la risorsa R_1 non rientrerà tra le risorse richieste da P_1.

Questo evidenzia il fatto come *non sia necessario,* per task aventi schema degli accessi *come quello associato in questo caso a P_i,* modellare gli scenari in cui essi rimangano in possesso di risorse utilizzate *in sezioni critiche più esterne* (vedi R_1 nell'esempio di sopra).

Tuttavia, tali equazioni e disequazioni sono ugualmente inserite nei modelli finali che consentono la stima del tempo di blocco B_{P1} in quanto *è altrettanto interessante avere un quadro informativo completo circa il comportamento di ciascun singolo task che concorre alla definizione del blocco del processo a maggior priorità P_1.*

L'utilità delle sezioni critiche pendenti può essere giustificata altresì dalla presenza nelle applicazioni r-t di processi P_i che accedono *simultaneamente* a due o più risorse, e rilasciano però queste stesse in istanti di tempo *differenti*.

Si consideri ad esempio il task set seguente:

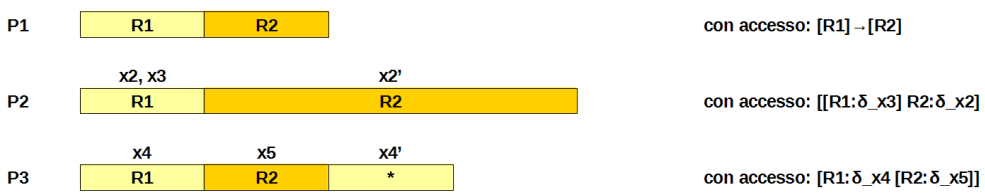


Figura 2.9 Esempio task set per utilizzo delle sezioni critiche pendenti

in cui è possibile osservare come:

1. il processo P_1, a priorità *massima,* richiede l'accesso in sequenza alle risorse R_1 ed R_2;
2. il processo P_2 acceda *simultaneamente* ad R_1 ed a R_2 secondo lo schema $[[R_1{:}\delta\, x_2]\ R_2{:}\delta\, x_3]$, quindi rilascia R_1 in anticipo rispetto ad R_2. Questo processo, dopo aver eseguito la sezione critica x_2, rimane in possesso del lock del semaforo R_2;
3. il processo P_3 acceda alle stesse risorse di cui sopra, ma nella classica modalità annidata $[R_1{:}\delta\, x_4\ [R_2{:}\delta\, x_5]]$.

Se fosse pertanto verificata la relazione "$\delta x_4 < (\delta x_2 + \delta x_3)$", è evidente che in ottica di massimizzazione del parametro B_{P1}, sarebbero scelte come bloccanti per il processo P_1:

- la sezione critica x_2, per quanto riguarda il possesso del lock della risorsa R_1;
- la sezione critica pendente $x_{3'}$ (cioè quella diminuita della durata temporale δx_2), per quanto concerne invece R_2;

rappresentando così lo scenario in cui P_2 rilascia per P_1 la risorsa R_1, rimanga in possesso di R_2, e successivamente liberi anche quest'ultima per lo stesso *processo a massima priorità*.

A questo punto, volendo riassumere i concetti chiave emersi nelle discussioni precedenti, è evidente come:

1. ciascuna combinazione z_k di P_i descriva specificatamente quale sia *l'insieme di risorse* $\{R_h, R_l, R_m, ...\}$ *di cui esso possieda già il lock*, e quindi per quali risorse *il processo* P_i *può causare potenzialmente un blocco diretto*;
2. ciascuna combinazione z_k di P_i descriva specificatamente quale sia l'insieme di risorse $\{R_p, R_q, R_r, ...\}$ di cui *esso non possiede il lock*, quindi per quali risorse il processo P_i *avrà la necessità di inoltrare una richiesta di liberazione del lock relativo*, subendo eventualmente un blocco diretto;
3. di un processo P_i potranno essere incluse nella soluzione ottima del problema ilp una o più combinazioni z_k, a patto che tale insieme di combinazioni z_k di P_i rappresentino una sequenza descritta da una delle catene di attivazione associate allo stesso processo.

Analizzando l'osservazione fatta la punto1 di sopra, allora per ogni processo P_i le combinazioni z_k potranno essere classificate a seconda che l'insieme delle sezioni critiche x_j relative a z_k possano rappresentare la *modalità con cui lo stesso processo* P_i *libera una delle risorse in suo possesso, in conseguenza ovviamente del fatto che sia stata inoltrata una richiesta di utilizzo da parte di un processo a priorità superiore* P_k.

E cioè, è possibile *ripartire* l'insieme delle combinazioni z_k di P_i come segue (vedi figura 2.10):

- *sottoinsieme*$_{1,Pi}$:= $\{z_1, z_2, z_3, z_4, z_5, z_6, z_7\}$, modellante lo scenario per cui *"P_i possiede il lock della risorsa R_1", nonché esso la rilascia attraverso l'esecuzione delle sezioni critiche della z_k che compare con valore unitario in soluzione ottima. La "selezione" di una z_k appartenente a questo sottoinsieme sarà possibile solo se è stata inoltrata una "richiesta di liberazione di R_1" da parte di un processo P_k con priorità superiore a P_i;*
- *sottoinsieme*$_{2,Pi}$:= $\{z_8, z_9, z_{10}, z_{11}, z_{12}\}$, modellante lo scenario per cui *"P_i possiede il lock della risorsa R_2", nonché esso la rilascia attraverso l'esecuzione delle sezioni critiche della z_k che compare con valore unitario in soluzione ottima. La "selezione" di una z_k appartenente a questo sottoinsieme sarà possibile solo se è stata inoltrata una "richiesta di liberazione di R_2" da parte di un processo P_k con priorità superiore a P_i;*
- *sottoinsieme*$_{3,Pi}$:= $\{z_{13}, z_{14}, z_{15}\}$, modellante lo scenario per cui *"P_i possiede il lock della risorsa R_3", nonché esso la rilascia attraverso l'esecuzione delle sezioni critiche della z_k che compare con valore unitario in soluzione ottima. La "selezione" di una z_k appartenente a questo sottoinsieme sarà possibile solo se è stata inoltrata una "richiesta di liberazione di R_3" da parte di un processo P_k con priorità superiore a P_i;*
- *sottoinsieme*$_{4,Pi}$:= $\{z_{16}\}$, modellante il caso per cui *"P_i possiede il lock della risorsa R_4", nonché esso la rilascia attraverso l'esecuzione delle sezioni critiche della z_k che compare con valore unitario in soluzione ottima. La "selezione" di una z_k appartenente a questo sottoinsieme sarà possibile solo se è stata inoltrata una "richiesta di liberazione di R_4" da parte di un processo P_k con priorità superiore a P_i.*

		R1	R2	R3	R4
sottoinsieme1,Pi	z1	1	0	0	0
	z2	1	0	0	0
	z3	1	0	0	0
	z4	1	0	0	0
	z5	1	0	0	0
	z6	1	0	0	0
	z7	1	0	0	0
sottoinsieme2,Pi	z8	0	1	0	0
	z9	0	1	0	0
	z10	0	1	0	0
	z11	0	1	0	0
	z12	0	1	0	0
sottoinsieme3,Pi	z13	0	0	1	0
	z14	0	0	1	0
	z15	0	0	1	0
sottoinsieme4,Pi	z16	0	0	0	1

Figura 2.10 Sottoinsiemi per le combinazioni z_k di P_i, compatibili con richieste di rilascio delle risorse $\{R_1,R_2,R_3,R_4\}$ inoltrate da un processo P_k con $p(P_k) > p(P_i)$

Per quanto riguarda il punto2 delle precedenti osservazioni, è facile comprendere come: qualora sia "selezionata" una z_k di P_i che preveda l'esecuzione di una sezione critica x_j con accesso annidato ad un dato insieme di risorse $\{R_x, R_y, \ldots\}$, allora è importante che per tale z_k *sia definito l'elenco ordinato delle risorse richieste* dal task.

Ad esempio, per il caso di P_i la tabella in figura 2.11 mette in rilievo come:

- se fosse presente "z_1=1" in soluzione ottima, allora il processo P_i inoltrerà necessariamente una richiesta per il possesso delle risorse $\{R_2, R_3$ ed $R_4\}$ (in quest'ordine);
- se fosse presente "z_2=1" o "z_8=1" in soluzione ottima, allora il processo P_i inoltrerà necessariamente una richiesta per il possesso delle risorse $\{R_3$ ed $R_4\}$ (in quest'ordine);
- se fosse presente "z_3=1", "z_9=1" o "z_{13}=1" in soluzione ottima, allora il processo P_i inoltrerà necessariamente una richiesta per il possesso di R_4;
- in relazione alla *"selezione" delle restanti* z_k, in questi casi alternativamente: o P_i avrà già preso possesso di tutte le risorse di cui necessita, oppure avrà addirittura rilasciato una o più di esse.

	R1	R2	R3	R4
z1	0	1	1	1
z2	0	0	1	1
z3	0	0	0	1
z4	0	0	0	0
z5	0	0	0	0
z6	0	0	0	0
z7	0	0	0	0
z8	0	0	1	1
z9	0	0	0	1
z10	0	0	0	0
z11	0	0	0	0
z12	0	0	0	0
z13	0	0	0	1
z14	0	0	0	0
z15	0	0	0	0
z16	0	0	0	0

Figura 2.11 Risorse richieste da P_i per "z_k=1" in soluzione ottima, con k ∈ {1, 2, 3, 8, 9, 13}

Queste due strutture tabellari si dimostrano necessarie ma non sufficienti a rappresentare in modo completo l'insieme delle proprietà che caratterizzano un dato processo e le sue modalità di esecuzione.

Per ciascuno task, infatti, è fondamentale costruire un'ultima tabella che mostri quali siano le risorse delle quali il processo detenga già il lock del relativo semaforo, a seconda ovviamente che esso sia assunto eseguito secondo la z_k "selezionata" (cioè tale che: "z_k=1") in soluzione ottima.

È facile notare in realtà come le informazioni presenti in tabella 2.12 siano perfettamente speculari a quelle descritte dalle celle della tabella in figura 2.11 nella pagina precedente.

	R1	R2	R3	R4
z1	1	0	0	0
z2	1	1	0	0
z3	1	1	1	0
z4	1	1	1	1
z5	1	1	1	0
z6	1	1	0	0
z7	1	0	0	0
z8	1	1	0	0
z9	1	1	1	0
z10	1	1	1	1
z11	1	1	1	0
z12	1	1	0	0
z13	1	1	1	0
z14	1	1	1	1
z15	1	1	1	0
z16	1	1	1	1

Figura 2.12 Risorse già in possesso di P_i in funzione della z_k selezionata in soluzione ottima.

Il motivo che giustifica la costruzione della tabella in questione, può essere facilmente spiegato analizzando il seguente esempio in cui si suppone che il Task Set di riferimento sia costituito dai processi $\{P_a,P_i,P_j\}$, con $p(P_a) > \{p(P_i),p(P_j)\}$, ed il Resource Set sia rappresentato dall'insieme $\{R_k,R_x,R_y\}$. A questo punto, si assume:

- che i due processi P_i e P_j prevedano l'accesso ed utilizzo delle risorse di cui sopra secondo lo schema $[R_k:\delta_{ik}\ [R_x:\delta_{ix}]]$ e $[R_k:\ \delta_{jk}\ [R_y:\delta_{jy}]]$ rispettivamente;
- che il processo P_a acceda alla risorsa R_k, quindi successivamente richieda R_y (non importa se questo secondo accesso sia o meno annidato rispetto al primo).

Con queste ipotesi, ed in funzione dell'estensione delle sezioni critiche dei processi P_i e P_j e del fatto che si voglia determinare il *massimo* tempo di blocco B_{Pa} di P_a, possono verificarsi due possibili scenari:

1. il processo P_a subisce un blocco diretto su R_k causato da P_i, e mai uno diretto da parte di P_j né per la risorsa R_k, né per quella R_y. E questo perché affinché P_j possa bloccare P_a su R_y, allora sarebbe necessario che P_j possegga anche la risorsa R_k, vista la struttura con annidamento prevista nel suo schema di accessi alle risorse. Ma questa è una condizione *non ammissibile* perché si è supposto che P_a abbia già subito un blocco su questa stessa risorsa da parte di P_i. Quindi P_a potrà subire un blocco su R_k o da parte di P_i o da parte di P_j;
2. se P_a subisse un blocco diretto da parte di P_j, questo sarebbe necessariamente sulla risorsa R_k e mai su quella R_y esattamente perché l'accesso a R_y è annidato rispetto a quello su R_k per quanto riguarda P_j. Quindi, nel momento in cui P_a inoltri la richiesta per il possesso di R_y, essa sarà stata già rilasciata da P_j, esattamente prima del termine della sua sezione

critica su R_k.

Al più sarà lo stesso P_j che subirà un blocco diretto su R_y se posseduta da *un ulteriore processo* P_g e $p(P_j) > p(P_g)$. Solo in questo caso P_a sperimenterà un ulteriore blocco, in particolare di tipo *transitivo*.

La schematizzazione proposta con la tabella mostrata in figura 2.12, permette quindi di tener traccia per ciascuna z_k delle risorse per le quali essa preveda un *lock già detenuto dal processo* P_i.

Inoltre, la tabella di figura 2.12 si dimostra di grande aiuto nella definizione dei vincoli lineari che evitino che nella soluzione ottima del problema ilp *ci sia una "selezione" di più* z_k *associate a processi differenti* del Task Set, e che *comportino però l'utilizzo, potenzialmente bloccante, di stesse risorse*.

A questo punto, per rappresentare opportunamente le proprietà del processo P_i schematizzate nelle tabelle precedenti, allora si considera fondamentale introdurre nel modello del task in esame un insieme di variabili binarie di *indicate come*:

- {R_x MANAGED BY P_i}, per ogni colonna che costituisce la tabella 2.12;
- {R_x REQUEST LAUNCHED BY P_i}, per ogni colonna della tabella 2.11 che possieda almeno una cella con un valore unitario;
- {R_x REQUEST MATCHED BY P_i}, per ogni sottoinsieme di z_k rappresentato in tabella 2.10;
- {P_i WILL USE R_x BY z_k}, per ogni *combinazione* z_k *che prevede l'esecuzione di sezioni critiche pendenti* x_j, *attraverso le quali il processo* P_i *rilascia la risorsa* R_x.

2.7 Utilizzo delle combinazioni z_k di P_i in Funzione Obiettivo

Il principio che porta a considerare come fondamentale l'utilizzo delle combinazioni z_k per rappresentare gli scenari esecutivi associati ai processi del task set in esame all'interno di un modello ilp, è il seguente:

- *"ciascuna* z_k *costituisce in realtà una variabile binaria del modello ilp, e la relativa selezione nella soluzione ottima (cioè il fatto che essa compaia in soluzione finale con valore unitario* z_k*=1) implica automaticamente che, in ottica di massimizzazione del tempo di blocco* B_{Pl}*, il processo* P_i *sarà considerato come "bloccante" per* P_l*, e le sezioni critiche di* P_i *che definiranno un contributo non nullo per* B_{Pl} *sono quelle associate alla sua o alle sue* z_k *selezionate"*.

Quanto appena detto, consente di affermare che:

- la Funzione Obiettivo del modello ilp per la stima del tempo di blocco B_{P1} del processo P_1, è descritta dalla massimizzazione della sommatoria delle combinazioni z_k associate ai processi P_i del task set di P_1, tali che essi ovviamente abbiano priorità inferiore a $p(P_1)$:
  - $F.O.:\ max \sum c_{zk} \cdot z_k;\ con\ z_k \in \{\ combinazioni\ di\ esecuzione\ di\ P_i \mid p(P_i) < p(P_1)\};$
- ogni combinazione z_k che compare in F.O., ha coefficiente c_{zk} definito dalla somma delle durate temporali $\delta(x_j)$ delle sezioni critiche x_j che si assumono eseguite per P_i quando "z_k=1" nella soluzione del problema ilp:
  - $c_{zk} = \sum \delta(x_j);\ con\ 'x_j = 1'\ per\ z_k.$

La schema logico alla base quindi della costruzione del modello ilp per ciascun processo che compone il Task Set di riferimento è il seguente:

1. per ogni processo P_i presente nell'applicazione r-t considerata, e con $p(P_i) < p(P_1)$ risulta necessario definire l'insieme delle combinazioni z_k che rappresentino le possibili modalità esecutive di P_i, e con cui quest'ultimo può contribuire al blocco di P_1;
2. per ogni z_k di P_i è necessario determinare i relativi coefficienti c_{zk} da porre in F.O;
3. infine, è necessario specificare l'insieme dei vincoli lineari che rappresentino opportunamente gli insiemi ammissibili ottenuti dalla selezione di una o più z_k per ciascun processo P_i, nonché compatibili con le catene di attivazione associate a tale processo.

2.8 Definizione dei vincoli lineari per le combinazioni z_k di P_i

Nei precedenti paragrafi si è discusso come per un dato processo P_i, possa essere selezionata in soluzione ottima una o più combinazioni z_k di esecuzione e, nel caso ci sia una *selezione multipla*, allora le sezioni critiche del processo in questione saranno eseguite *in modo discontinuo* e seguendo una delle *catene di attivazione definite per il processo in questione.*

Per descrivere le regole utili alla determinazione dei vincoli lineari che consentano a tal proposito una "selezione coerente" delle z_k, si prende nuovamente come esempio il processo P_i del caso di studio I e, dall'analisi della tabella in figura 2.8 precedente è possibile notare come questo task possa rimanere:

(a) in possesso delle risorse R_1, R_2 e R_3, se "selezionata" la z_{16} in soluzione ottima;

(b) in possesso delle risorse R_1 ed R_2, se "selezionata" una delle combinazioni $\{z_{13}, z_{14}, z_{15}\}$ in soluzione ottima;

(c) in possesso della risorse R_1, se "selezionata" una delle combinazioni $\{z_8, z_9, z_{10}, z_{11}, z_{12}\}$ in soluzione ottima.

Ed ancora, è possibile osservare come, se fosse verificata una delle condizioni (a), (b) o (c) e fosse inoltrata una richiesta di liberazione da parte di un processo P_k con $p(P_k) > p(P_i)$ per la risorsa:

1. R_1
   1.1. relativamente al caso (a), sarebbe necessario *"rendere selezionabile"* in soluzione ottima *anche* la combinazione z_5;
   1.2. per il caso (b), risulterebbe fondamentale *"rendere selezionabile"* in soluzione ottima *anche* la combinazione z_6;
   1.3. per il caso (c), si renderebbe necessario *"poter selezionare"* in soluzione finale anche la combinazione z_7;

2. R_2
   2.1. per il caso (a), sarebbe necessario *"rendere possibile la selezione"* in soluzione ottima della combinazione z_{11};
   2.2. per il caso (b), risulterebbe importante *"rendere possibile la selezione"* di z_{12};

3. R_3
   3.1. per il caso (a), sarebbe necessario "*rendere possibile la selezione"* anche della combinazione z_{15}.

Il primo vincolo pertanto da includere nel modello è il seguente:

$$v_1 : \sum_{i=1}^{16} z_i \leq 1 + P_i WILLUSER_1 BY z_7 + P_i WILLUSER_1 BY z_6 + P_i WILLUSER_1 BY z_5 + P_i WILLUSER_2 BY z_{12} + P_i WILLUSER_2 BY z_{11} + P_i WILLUSER_3 BY z_{15}$$

Per chiarire il significato di questa prima disequazione, si ricordi che:

- ciascuna delle variabili "*P_i WILL USE R_x BY z_k"*, rappresenta una *variabile binaria del modello ilp*, che sarà vincolata ad assumere il *valore "1"* ogni qualvolta sia supposta eseguita una z_j di P_i tale per cui questo stesso task rimanga in possesso della risorsa R_x, liberata attraverso la successiva esecuzione della combinazione z_k.
- per il caso del processo P_i, sono previste specificatamente (e ciò giustifica la loro presenza nel vincolo in esame):
  - $\{P_i$ WILL USE R_1 BY z_7, P_i WILL USE R_1 BY z_6, P_i WILL USE R_1 BY $z_5\}$, utili

rispettivamente per i casi 1.3, 1.2 e 1.1 di cui sopra;

- $\{P_i$ WILL USE R_2 BY z_{12}, P_i WILL USE R_2 BY $z_{11}\}$ utili rispettivamente utili per i casi 2.2 e 2.1 di cui sopra;
- $\{P_i$ WILL USE R_3 BY $z_{15}\}$ utile per il caso 3.1 precedente.

Il vincolo v_1 pertanto descrive come:

- per P_i sia possibile la selezione *di una sola delle sue combinazioni z_k quando tutte le variabili "P_i WILL USE R_x BY z_k" per esso definite saranno a valore logico "0"*;
- per P_i sia possibile la selezione *multipla delle sue combinazioni z_k* ogniqualvolta che *una o più variabili "P_i WILL USE R_x BY z_k" associate a P_i, risultino essere a valor logico "1"*.

La "selezione multipla di combinazioni z_k" per un dato processo P_i è di fondamentale importanza poiché consente ugualmente di rappresentare lo scenario "P_i resta in possesso della risorsa R_x", con la particolarità che la durata di z_k risulti inclusa nel computo finale del tempo di blocco B_1 di P_1, solamente se questa risorsa è effettivamente poi bloccante per P_1 o per un dato processo P_k a priorità superiore.

Si supponga ora che per P_i sia *selezionata in soluzione ottima la combinazione z_{16}*.

	x1	x2	x3	x4	x3'	x2'	x1'
z1	1	0	0	0	0	0	0
z2	0	1	0	0	0	0	1
z3	0	0	1	0	0	1	1
z4	0	0	0	1	1	1	1
z5	0	0	0	0	1	1	1
z6	0	0	0	0	0	1	1
z7	0	0	0	0	0	0	1
z8	0	1	0	0	0	0	0
z9	0	0	1	0	0	1	0
z10	0	0	0	1	1	1	0
z11	0	0	0	0	1	1	0
z12	0	0	0	0	0	1	0
z13	0	0	1	0	0	0	0
z14	0	0	0	1	1	0	0
z15	0	0	0	0	1	0	0

Figura 2.13 "z_j e z_y escluse" (vedi colorazioni: [ ] e [ ]) e "z_k da rendere selezionabili" (vedi: [ ], [ ] e [ ]) nell'ipotesi "$z_{16} = 1$ è presente in soluzione ottima"

In questo scenario, esaminando la tabella mostrata in pagina precedente, è possibile rilevare come:

(a) le *combinazioni $z_j \in \{z_1, z_2, z_3, z_8, z_9, z_{13}\}$ (vedi righe [ ])* non possano essere considerate come *"selezionabili con z_{16}"*, dato che esse rappresentano scenari in cui P_i sia supposto eseguito attraverso sezioni critiche esterne rispetto a quella x_4 associata a z_{16};

(b) le *combinazioni $z_y \in \{z_4, z_{10}, z_{14}\}$ (vedi righe [ ])* non siano compatibili con l'ipotesi di

"selezione multipla" in soluzione ottima con z_{16}, perché queste stesse prevedono l'utilizzo della risorsa R_4, la quale si ritiene utilizzata da P_i attraverso la z_{16};

(c) le *combinazioni* $z_k \in \{z_5, z_6, z_7, z_{11}, z_{12}, z_{15}\}$ *(vedi righe* [ ], [ ] e [ ]), rappresentano invece ulteriori combinazioni selezionabili in soluzione ottima per "z_{16}=1". Esse infatti, rientrano nel set delle *{z_k presenti nei nodi "figli" del diagramma ad albero di figura 2.5, e con nodo "radice" esattamente in z_{16}}*.

I nuovi vincoli da aggiungere al modello sono pertanto quelli che definiscono "*la mutua esclusione tra z_{16} e ciascuna z_j e z_y"* , appena indicate ai punti (a) e (b) di sopra. Quindi valgono le diseguaglianze:

$$v_2: z_{16} + z_1 \leq 1; \quad v_3: z_{16} + z_2 \leq 1; \quad v_4: z_{16} + z_3 \leq 1; \quad v_5: z_{16} + z_4 \leq 1;$$
$$v_6: z_{16} + z_8 \leq 1; \quad v_7: z_{16} + z_9 \leq 1; \quad v_8: z_{16} + z_{10} \leq 1;$$
$$v_9: z_{16} + z_{13} \leq 1; \quad v_{10}: z_{16} + z_{14} \leq 1$$

Le osservazioni appena discusse per lo scenario *"z_{16} assume valore unitario nella soluzione ottima"* devono essere estese anche per i casi in cui "*ciascuna delle restanti z_i, con i={1,..., 15}, sia inclusa nella soluzione con valore unitario"*. Lo schema da seguire per definire i corrispondenti vincoli lineari del modello, è il seguente:

a) data la generica z_i, $\forall z_j \notin$ {nodi del diagramma ad albero della catena di attivazione con radice in z_i} e con $j < i$;
b) $\forall$coppia (z_i, z_j): $\exists$vincolo "$z_i + z_j \leq 1$".

I vincoli lineari di *mutua esclusione* che si ricavano utilizzando questa regola sono:

per il sottoinsieme di combinazioni $\{z_{13}, z_{14}, z_{15}\}$

$$v_{11}: z_{15} + z_1 \leq 1; \quad v_{12}: z_{15} + z_2 \leq 1; \quad v_{13}: z_{15} + z_3 \leq 1; \quad v_{14}: z_{15} + z_4 \leq 1; \quad v_{15}: z_{15} + z_5 \leq 1; \quad v_{16}: z_{15} + z_8 \leq 1; \quad v_{17}: z_{15} + z_9 \leq 1; \quad v_{18}: z_{15} + z_{10} \leq 1; \quad v_{19}: z_{15} + z_{11} \leq 1;$$
$$v_{20}: z_{14} + z_1 \leq 1; \quad v_{21}: z_{14} + z_2 \leq 1; \quad v_{22}: z_{14} + z_3 \leq 1; \quad v_{23}: z_{14} + z_4 \leq 1; \quad v_{24}: z_{14} + z_5 \leq 1; \quad v_{25}: z_{14} + z_8 \leq 1; \quad v_{26}: z_{14} + z_9 \leq 1; \quad v_{27}: z_{14} + z_{10} \leq 1; \quad v_{28}: z_{14} + z_{11} \leq 1;$$
$$v_{29}: z_{13} + z_1 \leq 1; \quad v_{30}: z_{13} + z_2 \leq 1; \quad v_{31}: z_{13} + z_3 \leq 1; \quad v_{32}: z_{13} + z_4 \leq 1; \quad v_{33}: z_{13} + z_5 \leq 1; \quad v_{34}: z_{13} + z_8 \leq 1; \quad v_{35}: z_{13} + z_9 \leq 1; \quad v_{36}: z_{13} + z_{10} \leq 1; \quad v_{37}: z_{13} + z_{11} \leq 1;$$

per il sottoinsieme di combinazioni $\{z_8, z_9, z_{10}, z_{11}, z_{12}\}$

$v_{38}: z_{12} + z_1 \leq 1;$ $v_{44}: z_{11} + z_1 \leq 1;$ $v_{50}: z_{10} + z_1 \leq 1;$ $v_{56}: z_9 + z_1 \leq 1;$ $v_{62}: z_8 + z_1 \leq 1;$
$v_{39}: z_{12} + z_2 \leq 1;$ $v_{45}: z_{11} + z_2 \leq 1;$ $v_{51}: z_{10} + z_2 \leq 1;$ $v_{57}: z_9 + z_2 \leq 1;$ $v_{63}: z_8 + z_2 \leq 1;$
$v_{40}: z_{12} + z_3 \leq 1;$ $v_{46}: z_{11} + z_3 \leq 1;$ $v_{52}: z_{10} + z_3 \leq 1;$ $v_{58}: z_9 + z_3 \leq 1;$ $v_{64}: z_8 + z_3 \leq 1;$
$v_{41}: z_{12} + z_4 \leq 1;$ $v_{47}: z_{11} + z_4 \leq 1;$ $v_{53}: z_{10} + z_4 \leq 1;$ $v_{59}: z_9 + z_4 \leq 1;$ $v_{65}: z_8 + z_4 \leq 1;$
$v_{42}: z_{12} + z_5 \leq 1;$ $v_{48}: z_{11} + z_5 \leq 1;$ $v_{54}: z_{10} + z_5 \leq 1;$ $v_{60}: z_9 + z_5 \leq 1;$ $v_{66}: z_8 + z_5 \leq 1;$
$v_{43}: z_{12} + z_6 \leq 1;$ $v_{49}: z_{11} + z_6 \leq 1;$ $v_{55}: z_{10} + z_6 \leq 1;$ $v_{61}: z_9 + z_6 \leq 1;$ $v_{67}: z_8 + z_6 \leq 1;$

Inoltre, è importante rilevare come i ragionamenti appena fatti per le z_k appartenenti a $\{z_{16}\}$, $\{z_{15}, z_{14}, z_{13}\}$ ed a $\{z_{12}, \ldots, z_8\}$, non valgano per quanto riguarda le combinazioni di esecuzione appartenenti al sottoinsieme $\{z_1, z_2, z_3, z_4, z_5, z_6, z_7\}$. Per esse infatti non sarà definito alcuno dei vincoli lineari di sopra poiché è facile osservare come nessuna di tali combinazioni abbia *nodi figli* in uno qualsiasi dei diagrammi ad albero che rappresentano le catene di attivazione del processo P_i.

Per quanto riguarda la gestione delle variabili di tipo $\{P_i$ WILL USE R_x BY $z_k\}$, la strategia adottata è la seguente: quando tale variabile è vincolata ad assumere *valore unitario*, allora non si forza la corrispondente z_k a valere anch'essa "1". Bensì, si fa in modo che: "*tutte le combinazioni z_j che prevedano la liberazione del lock della medesima R_x, siano vincolate adottare valore nullo"* (Regola 1.2).

Per rendere facilmente comprensibile il ragionamento circa l'inopportunità di introdurre un vincolo del tipo "P_i *WILL USE* R_x *BY* $z_k = z_k$*"* nel modello di P_i, è sufficiente notare come: se per questo stesso processo valesse la relazione "P_i *WILL USE* R_x *BY* $z_k = 1$*"* , allora dall'equazione di prima:

- z_k sarebbe *"selezionata"* in soluzione ottima, anche allorché la R_x rilasciata attraverso la/le sezione/i critica/che associata/e alla stessa z_k non risultassero effettivamente bloccante/i per nessun processo P_k a priorità superiore;
- si otterrebbe dal processo di ottimizzazione, una stima errata del massimo tempo di blocco B_{P1} di P_1, poiché:
  (a) se per assurdo, il coefficiente di z_k in F.O. fosse *negativo* ($c\_z_k < 0$), allora tale misura sarebbe in realtà erroneamente *decrementata del valore di c_zk.*
  (b) se il coefficiente di z_k in F.O. fosse *positivo* ($c\_z_k > 0$), allora tale misura sarebbe in realtà erroneamente *incrementata del valore di c_zk.*

Utilizzando la Regola 1.2, è possibile definire interamente il set di vincoli associati alla gestione

della *variabile* $P_i\,WILL\,USE\,R_1\,BY\,z_7$ (vedi vincoli [68:73]), mostrati di sotto.

vincoli associati alla variabile binaria $P_i\,WILL\,USE\,R_1\,BY\,z_7$

$$v_{68}:\ z_1 + P_i\,WILL\,USE\,R_1\,BY\,z_7 \leq 1;$$
$$v_{69}:\ z_2 + P_i\,WILL\,USE\,R_1\,BY\,z_7 \leq 1;$$
$$v_{70}:\ z_3 + P_i\,WILL\,USE\,R_1\,BY\,z_7 \leq 1;$$
$$v_{71}:\ z_4 + P_i\,WILL\,USE\,R_1\,BY\,z_7 \leq 1;$$
$$v_{72}:\ z_5 + P_i\,WILL\,USE\,R_1\,BY\,z_7 \leq 1;$$
$$v_{73}:\ z_6 + P_i\,WILL\,USE\,R_1\,BY\,z_7 \leq 1;$$

Osservando ora nelle pagine successive di questo capitolo il complesso dei vincoli lineari definiti per la gestione di ciascuna delle variabili P_i WILL USE R_x BY z_k, allora è facile notare *come la Regola 1.2 sia sufficiente per determinare solamente le disequazioni*:

- $[v_{68} : v_{73}]$ di P_i WILL USE R_1 BY z_7;
- $[v_{74} : v_{79}]$ di P_i WILL USE R_1 BY z_6;
- $[v_{85} : v_{90}]$ di P_i WILL USE R_1 BY z_5;
- $[v_{118} : v_{121}]$ di P_i WILL USE R_2 BY z_{12};
- $[v_{102} : v_{105}]$ di P_i WILL USE R_2 BY _z_{11};
- $\{v_{131}, v_{132}\}$ di P_i WILL USE R_3 BY z_{15};

ma non quelli restanti associati alle variabili binarie P_i WILL USE R_x BY z_k.

Un'importante precisazione per la Regola 1.2 è infatti la seguente:

- *"ciascuna P_i_WILL_USE_R_x_BY_z_k, dev'essere utilizzata per escludere dalla soluzione ottima ogni combinazione z_j che non compare in nessuna delle catene di attivazione relative alla z_k resa "selezionabile" a causa della relazione "$P_i\,WILL\,USE\,R_x\,BY\,z_k = 1$"*.

Un esempio esplicativo di quanto detto, potrebbe essere il seguente: si consideri P_i WILL USE R_3 BY z_{15}, quindi si noti come in corrispondenza del valor logico "1" per questa variabile, la combinazione d'esecuzione che diviene *selezionabile* risulti essere la z_{15}.

A questo punto, esaminando la *catene di attivazione* per il processo P_i e relativamente a z_{15}, è evidente che:

- tra i *nodi di livello superiore* (cioè tra quelle per cui il nodo associato a z_{15} *è figlio*), è presente solamente la combinazione $\{z_{16}\}$;
- tra i *nodi di livello inferiore* (cioè che hanno come *nodo padre* z_{15}), siano presenti le combinazioni $\{z_{12}, z_7, z_6\}$.

Da queste considerazioni si deduce che la variabile P_i WILL USE R_3 BY z_{15} sarà adoperata per escludere (con espressione insiemistica): *{tutte le sezioni z_k di P_i} \ { {z_{16}} U {z_{12}, z_7, z_6} }*.
In conclusione, estendendo opportunamente il ragionamento appena fatto ai casi delle altre variabili di tipo {P_i WILL USE R_x BY z_k} è possibile *definire in modo completo i vincoli lineari ad esse associati*.

vincoli associati alla variabile binaria $P_i WILL\,USE\,R_1\,BY\,z_6$

$v_{74}: z_1 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{75}: z_2 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{76}: z_3 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{77}: z_4 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{78}: z_5 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{79}: z_7 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{80}: z_8 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{81}: z_9 + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{82}: z_{10} + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{83}: z_{11} + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$
$v_{84}: z_{12} + P_i WILL\,USE\,R_1\,BY\,z_6 \le 1;$

vincoli associati alla variabile binaria $P_i WILL\,USE\,R_1\,BY\,z_5$

$v_{85}: z_1 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{86}: z_2 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{87}: z_3 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{88}: z_4 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{89}: z_6 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{90}: z_7 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{91}: z_8 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{92}: z_9 + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{93}: z_{10} + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{94}: z_{11} + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{95}: z_{12} + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{96}: z_{13} + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{97}: z_{14} + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$
$v_{98}: z_{15} + P_i WILL\,USE\,R_1\,BY\,z_5 \le 1;$

vincoli associati alla variabile binaria $P_i WILL\,USE\,R_2\,BY\,z_{11}$

$v_{99}: z_{13} + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{100}: z_{14} + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{101}: z_{15} + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{102}: z_8 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{103}: z_9 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{104}: z_{10} + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{105}: z_{12} + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{106}: z_1 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{107}: z_2 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{108}: z_3 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{109}: z_4 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{110}: z_5 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$
$v_{111}: z_6 + P_i WILL\,USE\,R_2\,BY\,z_{11} \le 1;$

vincoli associati alla variabile binaria $P_i WILL\,USE\,R_2\,BY\,z_{12}$

$v_{112}: z_1 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{113}: z_2 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{114}: z_3 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{115}: z_4 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{116}: z_5 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{117}: z_6 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{118}: z_8 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{119}: z_9 + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{120}: z_{10} + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$
$v_{121}: z_{11} + P_i WILL\,USE\,R_2\,BY\,z_{12} \le 1;$

vincoli associati alla variabile binaria $P_i WILL\,USE\,R_3\,BY\,z_{15}$

$v_{122}: z_1 + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{123}: z_2 + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{124}: z_3 + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{125}: z_4 + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{126}: z_5 + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{127}: z_8 + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{128}: z_9 + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{129}: z_{10} + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{130}: z_{11} + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{131}: z_{13} + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$
$v_{132}: z_{14} + P_i WILL\,USE\,R_3\,BY\,z_{15} \le 1;$

A questo punto è lecito domandarsi quale sia *il criterio sulla base del quale si attribuisce valore non nullo* a ciascuna delle variabili di tipo $\{P_i \text{ WILL USE } R_x \text{ BY } z_k\}$.

Ancora una volta, si supponga che:

- *la combinazione z_j sia stata inclusa in soluzione ottima, e che il processo P_i rimanga in possesso della risorsa R_x;*
- *la stessa R_x sia quindi richiesta da un processo P_k, e liberata da P_i attraverso l'ulteriore "selezione" di z_k.*

È possibile constatare come la corrispondente variabile P_i WILL USE R_x BY z_k sarà necessariamente vincolata ad assumere valor logico "1" quando vale "z_j=1", tale che: tra le due combinazioni z_j e z_k esiste un arco *diretto* nel diagramma ad albero delle catene di attivazione, e dunque *z_j è rappresentato in un nodo "padre" per quello che riporta invece z_k.*

Il ragionamento è molto semplice da applicare se si prende nuovamente in considerazione la tabella 2.8.

Ad esempio, per la riga z_{16} si nota come in corrispondenza della sua "selezione" in soluzione ottima, P_i rimanga in possesso di $\{R_1, R_2 \text{ e } R_3\}$, liberabili dallo stesso a fronte di una corrispondente richiesta, ed attraverso rispettivamente la "selezione" delle sezioni z_5, z_{11} e z_{15}.

Tutto ciò giustifica *l'equazione definita al vincolo v_{133}*, in cui compare *la combinazione z_{16} assieme alle variabili $\{P_i \text{ WILL USE } R_1 \text{ BY } z_5,\ P_i \text{ WILL USE } R_2 \text{ BY } z_{11},\ P_i \text{ WILL USE } R_3 \text{ BY } z_{15}\}$*.

Un'importante precisazione riguarda il fatto che tra le variabili $\{P_i \text{ WILL USE } R_1 \text{ BY } z_5,\ P_i \text{ WILL USE } R_2 \text{ BY } z_{11},\ P_i \text{ WILL USE } R_3 \text{ BY } z_{15}\}$ sono definiti inoltre dei *vincoli lineari di mutua esclusione* (vedi vincoli [v134:v136]) poiché, data lo *schema con annidamenti dell'utilizzo che P_i fa di queste risorse*, e possibile dunque che esso *risponda solamente ad una sola tra le richieste di liberazione di {R1, R2, R3}.*

Il ragionamento è valido anche per il vincolo $\{v_{144}\}$, per quanto riguarda l'insieme $\{P_i \text{ WILL USE } R_1 \text{ BY } z_6,\ P_i \text{ WILL USE } R_2 \text{ BY } z_{12}\}$.

vincoli associati alla combinazione z_{16}

$$v_{133}: z_{16} - P_i \text{ WILL USE } R_1 \text{ BY } z_5 - P_i \text{ WILL USE } R_2 \text{ BY } z_{11} - P_i \text{ WILL USE } R_3 \text{ BY } z_{15} = 0;$$
$$v_{134}: P_i \text{ WILL USE } R_1 \text{ BY } z_5 + P_i \text{ WILL USE } R_2 \text{ BY } z_{11} \leq 1;$$
$$v_{135}: P_i \text{ WILL USE } R_1 \text{ BY } z_5 + P_i \text{ WILL USE } R_3 \text{ BY } z_{15} \leq 1;$$
$$v_{136}: P_i \text{ WILL USE } R_2 \text{ BY } z_{11} + P_i \text{ WILL USE } R_3 \text{ BY } z_{15} \leq 1;$$

Il caso appena analizzato circa la gestione delle variabili *di tipo P_i WILL USE R_x BY z_k* non è tuttavia esaustivo, poiché il valore assunto dalle variabili in questione può essere "controllato"

non considerando una singola z_k, ma quello di più di una purché appartenenti ad un set di combinazioni $\{z_g, z_h, z_i, \ldots\}$ con caratteristiche specifiche.

Si prenda in esame il caso del sottoinsieme$_{3,Pi}$:= $\{z_{13}, z_{14}, z_{15}\}$ e si noti come sia necessario che *una sola tra le variabili* $\{P_i \text{ WILL USE } R_2 \text{ BY } z_{12},\ P_i \text{ WILL USE } R_1 \text{ BY } z_6\}$ (vedi vincoli $\{v_{143}, v_{144}\}$) sia forzata ad assumere valore unitario se:

(1) una tra le combinazioni z_j del sottoinsieme $_{3,Pi}$ è "selezionata" in soluzione ottima;

(2) risulti essere a valor logico "1" almeno una tra le variabili P_i WILL USE R_x z_j, tale che z_j *appartenga al sottoinsieme* $_{3,Pi}$, in questo esempio: P_i WILL USE R_3 z_{15}.

L'importanza del punto2 deriva dal fatto che: se anche non fosse "selezionata" alcuna delle z_k appartenenti al sottoinsieme$_{3,Pi}$, è necessario ugualmente assumere che P_i sia in possesso delle risorse associate ad almeno una delle catene di attivazione eventualmente *innescate* dalla relazione "P_i WILL USE R_3 z_{15}=1".

Queste osservazioni giustifica le disequazioni [v_{138}:v_{142}] (esse definiscono un Or Logico tra le condizioni espresse ai punti1 e 2 precedenti) e l'equazione $\{v_{143}\}$.

Inoltre, è ovvia la conseguente estensione di tale ragionamento al caso della variabile P_i WILL USE R_1 z_7, "controllata" attraverso le combinazioni $\{z_8, \ldots, z_{12}\}$ e le variabili $\{P_i$ WILL USE R_2 z_{11}, P_i WILL USE R_2 $z_{12}\}$ (vincoli [v_{146}:v_{154}]).

In ultimo, nella definizione delle disequazioni ed equazioni del modello di P_i si è sinora ignorato il fatto che esistano più combinazioni z_k di P_i, la cui "selezione" in soluzione ottima implica che il processo P_i stia difatti *liberando* una determinata risorsa R_x del suo Resource Set in risposta ad un blocco diretto subito su di essa da parte di un processo P_k a priorità superiore.

È corretto ovviamente che P_i rilasci il lock del semaforo R_x, ma solamente attraverso una delle combinazioni z_k per cui esso risulti bloccante (relativamente sempre ad R_x).

Pertanto, riprendendo i *sottoinsiemi di* z_k già determinati e rappresentati attraverso la tabella in figura 2.10, è possibile definire i seguenti vincoli lineari di mutua esclusione:

- per le *combinazioni appartenenti al sottoinsieme* $_{1,Pi}$, vedi vincolo v_{155};
- per le *combinazioni appartenenti al sottoinsieme* $_{2,Pi}$, vedi vincolo v_{145};
- per le *combinazioni appartenenti al sottoinsieme* $_{3,Pi}$, vedi vincolo v_{137}.

Inoltre, questo ragionamento permette di precisare come: ogniqualvolta esista un insieme del tipo $\{P_i$ WILL USE R_x BY z_k, P_i WILL USE R_x BY z_j, P_i WILL USE R_x BY z_h, … $\}$ (nota: la *cardinalità* di tale set si suppone maggiore o uguale a 2), ed indicante per l'appunto che P_i possa di fatti utilizzare la risorsa R_x se "selezionata" in soluzione ottima una tra le combinazioni $\{z_k, z_j, z_h\}$, allora è necessario definire le *diseguaglianze di mutua esclusione anche tra variabili del set*

{P_i WILL USE R_x BY z_k, P_i WILL USE R_x BY z_j, P_i WILL USE R_x BY z_h, … }.

Per il caso del processo P_i, ciò giustifica specificatamente l'introduzione dei vincoli v_{156} e v_{157} nel modello, rispettivamente per gli insiemi {P_i WILL USE R_1 BY z_5, P_i WILL USE R_1 BY z_6, P_i WILL USE R_1 BY z_7} e {P_i WILL USE R_2 BY z_{11}, P_i WILL USE R_2 BY z_{12}}.

I vincoli lineari precedentemente menzionati, sono mostrati nella pagina seguente.

vincoli associati alle combinazioni $\{z_{13}, z_{14}, z_{15}\}$

$$v_{137}: z_{13} + z_{14} + z_{15} \leq 1;$$

$$v_{138}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_3\ REQUEST - z_{13} \geq 0;$$
$$v_{139}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_3\ REQUEST - z_{14} \geq 0;$$
$$v_{140}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_3\ REQUEST - z_{15} \geq 0;$$
$$v_{141}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_3\ REQUEST - P_i\ WILL\ USE\ R_3\ BY\ z_{15} \geq 0;$$
$$v_{142}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_3\ REQUEST - P_i\ WILL\ USE\ R_3\ BY\ z_{15} - z_{15} - z_{14} - z_{13} \leq 0;$$

$$v_{143}: P_i\ WILL\ USE\ R_1\ BY\ z_6 + P_i\ WILL\ USE\ R_2\ BY\ z_{12} - OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_3\ REQUEST = 0;$$

$$v_{144}: P_i\ WILL\ USE\ R_1\ BY\ z_6 + P_i\ WILL\ USE\ R_2\ BY\ z_{12} \leq 1;$$

vincoli associati alle combinazioni $\{z_8, z_9, z_{10}, z_{11}, z_{12}\}$

$$v_{145}: z_8 + z_9 + z_{10} + z_{11} + z_{12} \leq 1;$$

$$v_{146}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - z_8 \geq 0;$$
$$v_{147}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - z_9 \geq 0;$$
$$v_{148}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - z_{10} \geq 0;$$
$$v_{149}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - z_{11} \geq 0;$$
$$v_{150}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - z_{12} \geq 0;$$
$$v_{151}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - P_i\ WILL\ USE\ R_2\ BY\ z_{11} \geq 0;$$
$$v_{152}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - P_i\ WILL\ USE\ R_2\ BY\ z_{12} \geq 0;$$
$$v_{153}: OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST - P_i\ WILL\ USE\ R_2\ BY\ z_{11} - P_i\ WILL\ USE\ R_2\ BY\ z_{12} - z_8 - z_9 - z_{10} - z_{11} - z_{12} \leq 0;$$

$$v_{154}: P_i\ WILL\ USE\ R_1\ BY\ z_7 - OR\ RESULT\ FOR\ z_j s\ FEASIBLE\ FOR\ R_2\ REQUEST = 0;$$

vincolo associato alle combinazioni $\{z_1, z_2, z_3, z_4, z_5, z_6, z_7\}$

$$v_{155}: z_1 + z_2 + z_3 + z_4 + z_5 + z_6 + z_7 \leq 1;$$

vincolo per la mutua esclusione per le *variabili utili alla gestione delle sezioni critiche pendenti che prevedono l'utilizzo della risorsa* R_1

$$v_{156}: P_i WILL USE R_1 BY z_5 + P_i WILL USE R_1 BY z_6 + P_i WILL USE R_1 BY z_7 \leq 1;$$

vincolo per la mutua esclusione per le variabili utili alla gestione delle sezioni critiche pendenti che prevedono l'utilizzo della risorsa R_2

$$v_{157}: P_i WILL USE R_2 BY z_{11} + P_i WILL USE R_2 BY z_{12} \leq 1;$$

Un'ultima precisazione per il modello del processo P_i in esame riguarda i vincoli [v_{158}:v_{191}].

Per spiegare il loro ruolo nella rappresentazione del comportamento di questo processo, si suppone di analizzare a mo' di esempio le variabili {z_7 CANNOT BE ACTIVATED} e {z_8 z_9 z_{10} z_{11} z_{12} ALL NULL}.

In particolare, la variabile {z_8 z_9 z_{10} z_{11} z_{12} ALL NULL} è utile ad identificare lo scenario in cui

ciascuna combinazione z_j appartenente all'insieme $\{z_8, z_9, z_{10}, z_{11}, z_{12}\}$ compaia in soluzione ottima con *valore nullo*. In questo caso, risulta *inammissibile la possibilità che la sezione critica pendente* z_7 *possa essere "selezionata"*, anche qualora fosse assunta verificata l'ipotesi per cui P_i sia il task nelle possibilità di rilasciare la risorsa R_1 attraverso la stessa z_7 (cioè si suppone verificata la relazione "P_i WILL USE R_1 BY z_7=1").

L'attivazione di $\{P_i$ WILL USE R_1 BY $z_7\}$ infatti, in contemporaneità con quella $\{z_8\, z_9\, z_{10}\, z_{11}\, z_{12}$ ALL NULL$\}$, rappresenta lo scenario per cui si assume che P_i possieda ugualmente il lock della risorsa R_1, ma che essa non sia stato rilasciata in quanto *non selezionata in soluzione ottima nessuna delle combinazione di esecuzione* z_k *che compare come "nodo padre" di* z_7*, in una della catena di attivazione di* P_i*.*

In questo contesto sarà altresì "attivata" la variabile "z_7 CANNOT BE ACTIVATED = 1", così da poter escludere "z_7=1" dalla soluzione finale del problema ilp.

Pertanto, la *sezione critica pendente* z_7 *potrà assumere valore unitario* in soluzione finale se e solo se risultano verificate *insieme* le seguenti condizioni:

- è attiva una combinazione z_j che rappresenti una modalità esecutiva di P_i che permetta l'attivazione di z_7, cioè risulta essere a valor unitario una delle z_j del set $\{z_8, z_9, z_{10}, z_{11}, z_{12}\}$.
- è a *valor unitario la variabile binaria che modella il rilascio di* R_1 *per mezzo di* z_7, cioè "P_i WILL USE R_1 BY z_7".

Questo discorso può essere esteso anche alle variabili:

1. $\{z_6$ CANNOT BE ACTIVATED, z_{12} CANNOT BE ACTIVATED$\}$, utilizzate in associazione con quella $\{z_{13}\, z_{14}\, z_{15}$ ALL NULL$\}$;
2. $\{z_5$ CANNOT BE ACTIVATED, z_{11} CANNOT BE ACTIVATED, z_{15} CANNOT BE ACTIVATED$\}$ utilizzate assieme alla variabile di esecuzione z_{16} (nota: in questo caso non è necessario definire una variabile di tipo "z_a z_b ..ALL NULL", dato che l'unica combinazione di esecuzione che può attivare $\{z_5, z_{11}$ e $z_{15}\}$ è effettivamente quella z_{16}).

Nella pagina seguente sono mostrate le relazioni di modello associate ai ragionamenti appena fatti.

vincoli per la gestione della sezione critica z_7
con quelle $\{ z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL,\ z_7\_CANNOT\_BE\_ACTIVATED \}$

$$v_{158}: z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL + z_8 \leq 1;$$
$$v_{159}: z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL + z_9 \leq 1;$$
$$v_{160}: z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL + z_{10} \leq 1;$$
$$v_{161}: z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL + z_{11} \leq 1;$$
$$v_{162}: z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL + z_{12} \leq 1;$$
$$v_{163}: z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL + z_{12} + z_{11} + z_{10} + z_9 + z_8 \geq 1;$$

$$v_{164}: z_7\_CANNOT\_BE\_ACTIVATED - z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL \leq 0;$$
$$v_{165}: z_7\_CANNOT\_BE\_ACTIVATED - P_i\_WILL\,USE\_R_1\_BY\_z_7 \leq 0;$$
$$v_{166}: z_7\_CANNOT\_BE\_ACTIVATED - z_8\_z_9\_z_{10}\_z_{11}\_z_{12}\_ALL\_NULL - P_i\_WILL\,USE\_R_1\_BY\_z_7 \geq -1;$$

$$v_{167}: z_7\_CANNOT\_BE\_ACTIVATED + z_7 \leq 1;$$

vincoli per la gestione della sezioni critiche $\{z_6,\ z_{12}\}$
con quelle $\{ z_{13}\_z_{14}\_z_{15}\_ALL\_NULL,\ z_6\_CANNOT\_BE\_ACTIVATED,\ z_{12}\_CANNOT\_BE\_ACTIVATED \}$

$$v_{168}: z_{13}\_z_{14}\_z_{15}\_NULL + z_{13} \leq 1;$$
$$v_{169}: z_{13}\_z_{14}\_z_{15}\_NULL + z_{14} \leq 1;$$
$$v_{170}: z_{13}\_z_{14}\_z_{15}\_NULL + z_{15} \leq 1;$$
$$v_{171}: z_{13}\_z_{14}\_z_{15}\_NULL + z_{15} + z_{14} + z_{13} \geq 1;$$

$$v_{172}: z_6\_CANNOT\_BE\_ACTIVATED - z_{13}\_z_{14}\_z_{15}\_ALL\_NULL \leq 0;$$
$$v_{173}: z_6\_CANNOT\_BE\_ACTIVATED - P_i\_WILL\,USE\_R_1\_BY\_z_6 \leq 0;$$
$$v_{174}: z_6\_CANNOT\_BE\_ACTIVATED - z_{13}\_z_{14}\_z_{15}\_ALL\_NULL - P_i\_WILL\,USE\_R_1\_BY\_z_6 \geq -1;$$

$$v_{175}: z_6\_CANNOT\_BE\_ACTIVATED + z_6 \leq 1;$$

$$v_{176}: z_{12}\_CANNOT\_BE\_ACTIVATED - z_{13}\_z_{14}\_z_{15}\_ALL\_NULL \leq 0;$$
$$v_{177}: z_{12}\_CANNOT\_BE\_ACTIVATED - P_i\_WILL\,USE\_R_2\_BY\_z_{12} \leq 0;$$
$$v_{178}: z_{12}\_CANNOT\_BE\_ACTIVATED - z_{13}\_z_{14}\_z_{15}\_ALL\_NULL - P_i\_WILL\,USE\_R_2\_BY\_z_{12} \geq -1;$$

$$v_{179}: z_{12}\_CANNOT\_BE\_ACTIVATED + z_{12} \leq 1;$$

vincoli per la gestione delle sezioni critiche $\{z_5,\ z_{11},\ z_{15}\}$
con quelle $\{ z_{16},\ z_5\_CANNOT\_BE\_ACTIVATED,\ z_{11}\_CANNOT\_BE\_ACTIVATED,\ z_{15}\_CANNOT\_BE\_ACTIVATED \}$

$$v_{180}: z_5\_CANNOT\_BE\_ACTIVATED - z_{16} \leq 0;$$
$$v_{181}: z_5\_CANNOT\_BE\_ACTIVATED - P_i\_WILL\,USE\_R_1\_BY\_z_5 \leq 0;$$
$$v_{182}: z_5\_CANNOT\_BE\_ACTIVATED - z_{16} - P_i\_WILL\,USE\_R_1\_BY\_z_5 \geq -1;$$

$$v_{183}: z_5\_CANNOT\_BE\_ACTIVATED + z_5 \leq 1;$$

$$v_{184}: z_{11}\_CANNOT\_BE\_ACTIVATED - z_{16} \leq 0;$$
$$v_{185}: z_{11}\_CANNOT\_BE\_ACTIVATED - P_i\_WILL\,USE\_R_2\_BY\_z_{11} \leq 0;$$
$$v_{186}: z_{11}\_CANNOT\_BE\_ACTIVATED - z_{16} - P_i\_WILL\,USE\_R_2\_BY\_z_{11} \geq -1;$$

$$v_{187}: z_{11}\_CANNOT\_BE\_ACTIVATED + z_{11} \leq 1;$$

$$v_{188}: z_{15}\_CANNOT\_BE\_ACTIVATED - z_{16} \leq 0;$$
$$v_{189}: z_{15}\_CANNOT\_BE\_ACTIVATED - P_i\_WILL\,USE\_R_3\_BY\_z_{15} \leq 0;$$
$$v_{190}: z_{15}\_CANNOT\_BE\_ACTIVATED - z_{16} - P_i\_WILL\,USE\_R_3\_BY\_z_{15} \geq -1;$$

$$v_{191}: z_{15}\_CANNOT\_BE\_ACTIVATED + z_{15} \leq 1;$$

L'attuale capitolo prosegue con l'analisi dei modelli dei due ulteriori casi di studio, più semplici rispetto a quello appena esaminato, per consentire una migliore comprensione delle procedure

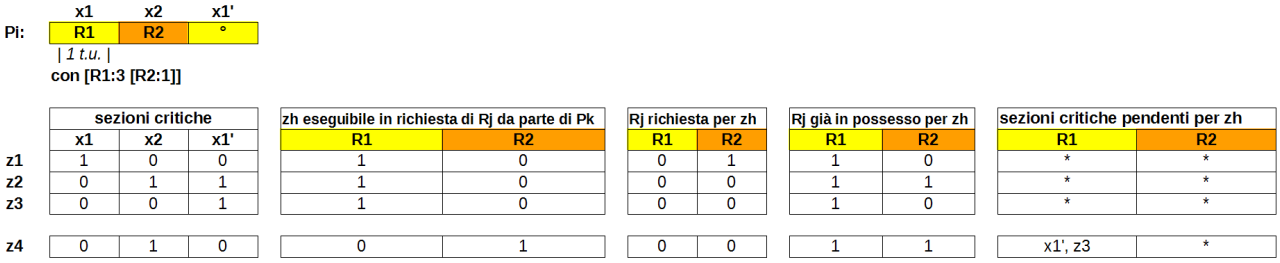
con cui si è scelto di rappresentare in un modello ilp i possibili task di applicazioni r-t.

Tali esempi in particolare assumono notevole rilevanza poiché compaiono nelle successive analisi dimostrative delle tecniche con cui è simulata la presenza di blocchi tra i processi delle applicazioni.

2.9 Analisi delle caratteristiche e modello ILP del task P_i per il caso di studio II

Nel caso di studio II è supposto che il generico processo P_i acceda alle risorse R_1 e R_2 in modalità annidata, e cioè secondo lo schema [R1:3 [R2:1]] mostrato nella figura 2.14.

In essa sono rappresentate anche le *tabelle delle proprietà di processo* che riassumono tutte le sue informazioni significative relative al task in esame.



	sezioni critiche			zh eseguibile in richiesta di Rj da parte di Pk		Rj richiesta per zh		Rj già in possesso per zh		sezioni critiche pendenti per zh	
	x1	x2	x1'	R1	R2	R1	R2	R1	R2	R1	R2
z1	1	0	0	1	0	0	1	1	0	*	*
z2	0	1	1	1	0	0	0	1	1	*	*
z3	0	0	1	1	0	0	0	1	0	*	*
z4	0	1	0	0	1	0	0	1	1	x1', z3	*

Figura 2.14 Schema degli accessi di P_i e relative tabelle delle proprietà di processo

Le precisazioni di maggior importanza riguardo il modello di P_i sono:

1. è facile in primo luogo notare come P_i possa rimanere in possesso della risorsa R_1 dopo l'esecuzione della sezione critica x_2, quindi si rende necessaria l'introduzione di una sola variabile binaria, indicata come P_i WILL USE R_1 BY z_3, fondamentale per la corretta gestione dell'unica catena di attivazione attribuibile a questo processo, ed il cui diagramma è mostrato di sotto.

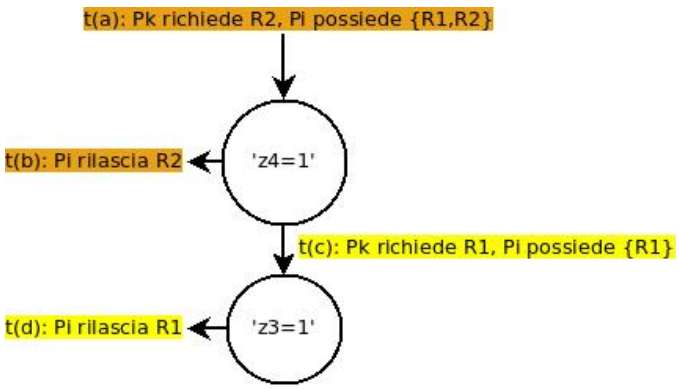


Figura 2.15 Catena di attivazione del processo P_i

2. nella rappresentazione del processo P_i sono introdotti per la prima volta le equazioni e le disequazioni lineari (vedi [v_8:v_{16}] del modello ILP completo nella pagina seguente) che determinano il valore finale assunto dalle variabili binarie {R_1 REQUEST MATCHED BY P_i, R_2 REQUEST MATCHED BY P_i, R_2 REQUEST LAUNCHED BY P_i} e {R_1_MANAGED_BY_P_i, R_2_MANAGED_BY_P_i}. Queste relazioni indicano in corrispondenza di quali combinazioni di esecuzione z_k il processo P_i rispettivamente: risponda ad una richiesta di liberazione di R_1 e/o R_2; richieda l'inoltro di una richiesta per il possesso del lock del semaforo di R_2; e per quali altre z_j si assume che questo task disponga già del possesso di R_1 e/o R_2.

Il modello ilp completo utilizzato per rappresentare il processo P_i è mostrato di sotto e nella pagina seguente.

vincoli per la modellazione del processo P_i

$$v_1: z_1 + z_2 + z_3 + z_4 - P_i\_WILL\_USE\_R_1\_BY\_z_3 \leq 1;$$

vincoli di mutua esclusione tra $\{ z_{21} , z_{22} , z_{23} \}$ *:*

$$v_2: z_1 + z_2 + z_3 \leq 1;$$

vincoli di mutua esclusione tra z_4 *e le combinazioni* $z_j \in \{ z_1 , z_2 , z_3 \}$ *che rispondono alla richiesta di* R_1 *e equazione di definizione della variabile* $P_i\_WILL\_USE\_R_1\_BY\_z_3$ *:*

$$v_3: z_4 - P_i\_WILL\_USE\_R_1\_BY\_z_3 = 0;$$
$$v_4: z_1 + z_4 \leq 1;$$
$$v_5: z_2 + z_4 \leq 1;$$

vincoli di mutua esclusione tra $\{z_1, z_2\}$ *e la variabile* $P_i\_WILL\_USE\_R_1\_BY\_z_3$ *:*

$$v_6: z_1 + P_i\_WILL\_USE\_R_1\_BY\_z_3 \leq 1;$$
$$v_7: z_2 + P_i\_WILL\_USE\_R_1\_BY\_z_3 \leq 1;$$

vincoli di definizione delle variabili $\{ R_{1,2}\_REQUEST\_MATCHED\_BY\_P_i \}$ *:*

$$v_8: R_1\_REQUEST\_MATCHED\_BY\_P_i - z_1 - z_2 - z_3 = 0;$$
$$v_9: R_2\_REQUEST\_MATCHED\_BY\_P_i - z_4 = 0;$$

equazione di definizione della variabile $\{ R_2\_REQUEST\_LAUNCHED\_BY\_P_i \}$ *:*

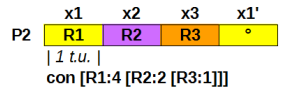
$$v_{10}: R_2\_REQUEST\_LAUNCHED\_BY\_P_i - z_1 = 0;$$

$$\textit{vincoli di definizione delle variabili } \{ R_{1,2}\_MANAGED\_BY\_P_i \} :$$
$$v_{11}: R_1\_MANAGED\_BY\_P_i - z_1 \geq 0;$$
$$v_{12}: R_1\_MANAGED\_BY\_P_i - z_2 \geq 0;$$
$$v_{13}: R_1\_MANAGED\_BY\_P_i - z_3 \geq 0;$$
$$v_{14}: R_1\_MANAGED\_BY\_P_i - z_4 \geq 0;$$
$$v_{15}: R_1\_MANAGED\_BY\_P_i - z_1 - z_2 - z_3 - z_4 \leq 0;$$

$$v_{16}: R_2\_MANAGED\_BY\_P_i - z_2 - z_4 = 0 .$$

2.10 Analisi delle caratteristiche e modello ILP del task P_i per il caso di studio III

Il terzo esempio notevole che si vuole indagare è quello in cui il generico task P_i acceda *in modalità annidata alle risorse del set {R_1, R_2, R_3}, secondo lo schema [R_1:4 [R_2:2 [R_3:1]]]* mostrato in figura 2.15. In essa sono riportate anche le principali caratteristiche di P_i attraverso le corrispondenti *tabelle delle proprietà di processo*.



	sezioni critiche incluse per zh				zh eseguibile in richiesta di Rj da parte di Pk			Rj richiesta per zh			Rj già in possesso per zh			sezioni critiche pendenti per zh		
	x1	x2	x3	x1'	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
z21	1	0	0	0	1	0	0	0	1	1	1	0	0	*	*	*
z22	0	1	0	1	1	0	0	0	0	1	1	1	0	*	*	*
z23	0	0	1	1	1	0	0	0	0	0	1	1	1	*	*	*
z24	0	0	0	1	1	0	0	0	0	0	1	0	0	*	*	*
z25	0	1	0	0	0	1	0	0	0	1	1	1	0	x1', z24	*	*
z26a	0	0	1	0	0	1	0	0	0	0	1	1	1	x1', z24	*	*
z26b	0	0	1	0	0	0	1	0	0	0	1	1	1	x1', z24	*	*

Figura 2.16 Schema degli accessi del task P_2 e relative tabelle delle proprietà di processo

Una breve osservazione, riguarda la simbologia introdotta nella immagine di sopra. E cioè: la ragione che spinge ad indicare il generico processo P_i come *quello P_2* è riconducibile al fatto che esso compaia come *processo a priorità p_2* (minore di p(P_1) ≡ priorità $_{MAX}$) in uno dei task set del successivo capitolo adoperati per dimostrare l'efficacia dei modelli ideati nel tener conto delle possibili interazioni bloccanti tra processi.

Inoltre, dalla figura precedente è facile notare come per elencare le combinazioni di esecuzione z_k è utilizzato un doppio pedice, tale che, se si prendesse in considerazione l'elemento z_{ij} allora:

- il *primo elemento "i" del pedice*, indica che tale combinazione z_{ij} appartiene al set di z_k definite per simulare l'esecuzione del processo P_i;
- il *secondo elemento "j" del pedice*, è utilizzato per ordinare *internamente* le combinazione di esecuzione definite per P_i.

Affinché sia sufficientemente chiaro il significato di ciascuno dei vincoli del modello ILP di P_2,

si rende necessario fare le seguenti precisazioni:

1. siccome gli istanti di rilascio per le risorse R_2 e R_3 coincidono, allora il processo P_2 può restare difatti in possesso della risorsa R_1 al termine dell'esecuzione di x_2 e di x_3. È necessario pertanto introdurre la variabile binaria P_i WILL USE R_1 BY z_{24} per modellare opportunamente tale scenario.
   La catena di attivazione di P_2 è rappresentata in figura 2.16 nella pagina seguente;

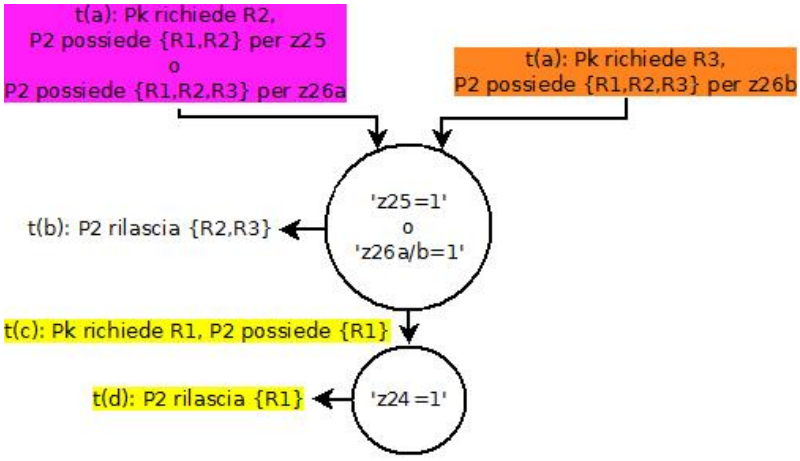


Figura 2.17 Catena di attivazione del processo P_2

2. nell'ipotesi che P_2 sia già in possesso delle risorse $\{R_1,R_2\}$ e abbia consumato per una frazione di tempo trascurabile la sezione critica x_3, quindi possegga già il lock del semaforo relativo ad R_3, allora l'esecuzione di x_3 per P_2 potrà avvenire sia a fronte di una *richiesta di liberazione della risorsa R_2*, sia in corrispondenza di una *richiesta per il possesso del lock di R_3.*
   Ciò giustifica la presenza per P_2 di due combinazioni di esecuzione z_{26a} e z_{26b} che prevedono il *consumo della medesima sezione critica x_3,* a fronte però rispettivamente dell'*attivazione delle variabili "R_2 REQUEST MATCHED BY P_2" e "R_3 REQUEST MATCHED BY P_2",* indicanti che P_2 sia assunto come il task che rilascia R_2 o R_3 tra quelli che compongono l'applicazione.
   Inoltre, anche se le combinazioni z_{26a} e z_{26b} *rispondono* a richieste di liberazione per risorse differenti, poiché prevedono l'esecuzione della medesima sezione critica x_3, allora per esse sarà necessario un relativo vincolo di mutua esclusione.
   In realtà, in tale vincolo è necessario includere anche la combinazione z_{25} poiché anch'essa prevede l'esecuzione della sezione critica x_3. È infatti assolutamente

deprecabile una soluzione finale del tipo (z_{25}=1; z_{26a}=1; z_{26b}=0) o (z_{25}=1; z_{26a}=0; z_{26b}=1). Tutto ciò giustifica l'introduzione della disequazione v_{14}.

Il modello *completo* del processo P_2 è mostrato di seguito.

vincoli per la modellazione del processo P_2

$$v_1: z_{21} + z_{22} + z_{23} + z_{24} + z_{25} + z_{26a} + z_{26b} - P_2\_WILL\_USE\_R_1\_BY\_z_{24} \leq 1;$$

vincoli di mutua esclusione tra $\{ z_{25}, z_{26a}, z_{26b} \}$
e le combinazioni $z_j \in \{ z_{21}, z_{22}, z_{23} \}$ *che rispondono alle richieste di* R_1 :

$$v_2: z_{25} + z_{21} \leq 1;$$
$$v_3: z_{25} + z_{22} \leq 1;$$
$$v_4: z_{25} + z_{23} \leq 1;$$
$$v_5: z_{26a} + z_{21} \leq 1;$$
$$v_6: z_{26a} + z_{22} \leq 1;$$
$$v_7: z_{26a} + z_{23} \leq 1;$$
$$v_8: z_{26b} + z_{21} \leq 1;$$
$$v_9: z_{26b} + z_{22} \leq 1;$$
$$v_{10}: z_{26b} + z_{23} \leq 1;$$

vincoli di mutua esclusione associati alla variabile $P_2\_WILL\_USE\_R_1\_BY\_z_{24}$:

$$v_{11}: z_{21} + P_2\_WILL\_USE\_R_1\_BY\_z_{24} \leq 1;$$
$$v_{12}: z_{22} + P_2\_WILL\_USE\_R_1\_BY\_z_{24} \leq 1;$$
$$v_{13}: z_{23} + P_2\_WILL\_USE\_R_1\_BY\_z_{24} \leq 1;$$

vincoli di mutua esclusione tra $\{ z_{25}, z_{26a}, z_{26b} \}$
ed equazione di definizione della variabile $P_2\_WILL\_USE\_R_1\_BY\_z_{24}$:

$$v_{14}: z_{25} + z_{26a} + z_{26b} \leq 1;$$
$$v_{15}: z_{25} + z_{26a} + z_{26b} - P_2\_WILL\_USE\_R_1\_BY\_z_{24} = 0;$$

vincoli di mutua esclusione tra $\{ z_{21}, z_{22}, z_{23}, z_{24} \}$:

$$v_{16}: z_{21} + z_{22} + z_{23} + z_{24} \leq 1;$$

vincoli per la definizione delle variabili $\{ R_{1,2,3}\_REQUEST\_MATCHED\_BY\_P_2 \}$:

$$v_{17}: R_1\_REQUEST\_MATCHED\_BY\_P_2 - z_{21} - z_{22} - z_{23} - z_{24} = 0;$$
$$v_{18}: R_2\_REQUEST\_MATCHED\_BY\_P_2 - z_{25} - z_{26a} = 0;$$
$$v_{19}: R_3\_REQUEST\_MATCHED\_BY\_P_2 - z_{26b} = 0;$$

vincoli per la definizione della variabile $R_1\_MANAGED\_BY\_P_2$:

v_{20}: $R_1\_MANAGED\_BY\_P_2 - z_{21} \geq 0$;

v_{21}: $R_1\_MANAGED\_BY\_P_2 - z_{22} \geq 0$;

v_{22}: $R_1\_MANAGED\_BY\_P_2 - z_{23} \geq 0$;

v_{23}: $R_1\_MANAGED\_BY\_P_2 - z_{24} \geq 0$;

v_{24}: $R_1\_MANAGED\_BY\_P_2 - z_{25} \geq 0$;

v_{25}: $R_1\_MANAGED\_BY\_P_2 - z_{26a} \geq 0$;

v_{26}: $R_1\_MANAGED\_BY\_P_2 - z_{26b} \geq 0$;

v_{27}: $R_1\_MANAGED\_BY\_P_2 - z_{21} - z_{22} - z_{23} - z_{24} - z_{25} - z_{26a} - z_{26b} \leq 0$;

vincoli per la definizione della variabile $R_2\_MANAGED\_BY\_P_2$:

v_{28}: $R_2\_MANAGED\_BY\_P_2 - z_{22} \geq 0$;

v_{29}: $R_2\_MANAGED\_BY\_P_2 - z_{23} \geq 0$;

v_{30}: $R_2\_MANAGED\_BY\_P_2 - z_{25} \geq 0$;

v_{31}: $R_2\_MANAGED\_BY\_P_2 - z_{26a} \geq 0$;

v_{32}: $R_2\_MANAGED\_BY\_P_2 - z_{26b} \geq 0$;

v_{33}: $R_2\_MANAGED\_BY\_P_2 - z_{22} - z_{23} - z_{25} - z_{26a} - z_{26b} \leq 0$;

vincoli per la definizione della variabile $R_3\_MANAGED\_BY\_P_2$:

v_{34}: $R_3\_MANAGED\_BY\_P_2 - z_{23} \geq 0$;

v_{35}: $R_3\_MANAGED\_BY\_P_2 - z_{26a} \geq 0$;

v_{36}: $R_3\_MANAGED\_BY\_P_2 - z_{26b} \geq 0$;

v_{37}: $R_3\_MANAGED\_BY\_P_2 - z_{23} - z_{26a} - z_{26b} \leq 0$;

vincoli per la definizione della variabile $R_2\_REQUEST\_LAUNCHED\_BY\_P_2$:

v_{38}: $R_2\_REQUEST\_LAUNCHED\_BY\_P_2 - z_{21} = 0$;

vincoli per la definizione della variabile $R_3\_REQUEST\_LAUNCHED\_BY\_P_2$:

v_{39}: $R_3\_REQUEST\_LAUNCHED\_BY\_P_2 - z_{21} - z_{22} - z_{25} = 0$;

2.11 Modelli ILP per task aventi più sezioni critiche esterne

È possibile che un'applicazione r-t includa *task aventi più sezioni critiche "esterne",* con *possibili ulteriori accessi annidati* in esse, come per il caso evidenziato dalla figura di sotto mostrante lo schema del generico processo P_k con tali caratteristiche.

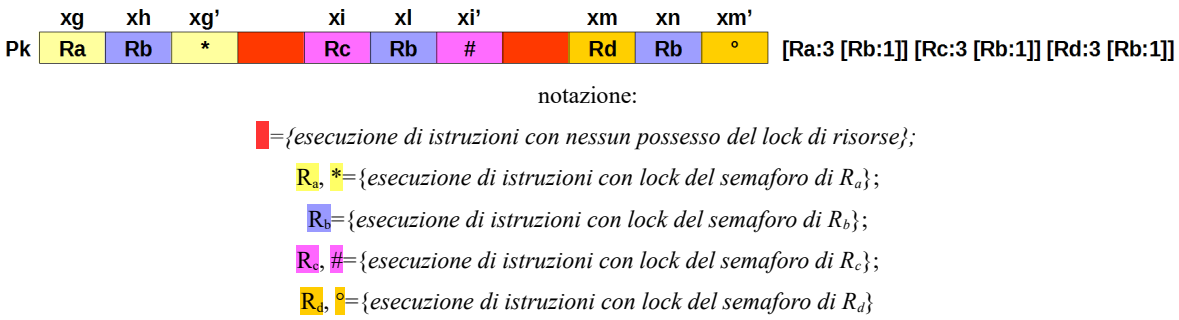


Figura 2.18 Schema degli accessi alle risorse del task P_k

Si rende pertanto necessario definire un insieme di regole attraverso le quali sia possibile

rappresentare, in un modello ILP di un'applicazione r-t, i task che abbiano questa particolare struttura.

2.12 Sezioni critiche esterne di P_k e task fittizi associati

In primo luogo è possibile osservare come questo processo possieda:

- tre sezioni critiche "esterne" $\{x_g, x_i, x_m\}$,rispettivamente associate all'utilizzo di $\{R_a, R_c, R_d\}$;
- ogni sezione di cui sopra prevede *l'accesso annidato* alla risorsa R_b (vedi: $\{x_h, x_l, x_n\}$).

I primi *step* da seguire per la modellazione di un processo con tali caratteristiche, sono:

a) *si considera* singolarmente *ogni sezione critica esterna* del processo;
b) *ciascuna di essa (e le eventuali sezioni annidate presenti),* sono rappresentate mediante l'introduzione di nuovo processo *fittizio* nel Task Set.

Applicando questo ragionamento a P_k, allora:

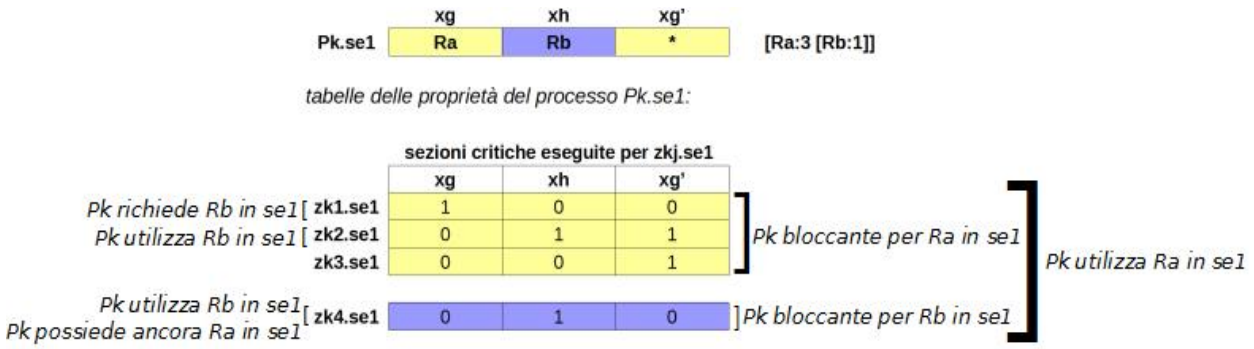
1. la sezione critica esterna x_g, e quella relativa annidata x_h, è rappresentata attraverso un *nuovo processo* $P_k.se1$;
2. la sezione critica esterna x_i, e quella relativa annidata x_l, è rappresentata attraverso un *nuovo processo* $P_k.se2$;
3. la sezione critica esterna x_m, e quella relativa annidata x_n, è rappresentata attraverso un *nuovo processo* $P_k.se3$.

Una piccola osservazione riguarda *l'acronimo "*se_i*"* presente per ogni *nuovo task introdotto*. Esso ha il significato di *"sezione critica esterna i-esima"*, ed indica che il processo $P_k.se_i$ rappresenta il comportamento di P_k qualora esso *fosse bloccante attraverso la propria sezione critica esterna i- -esima*, o eventualmente attraverso *quelle annidate in* se_i.

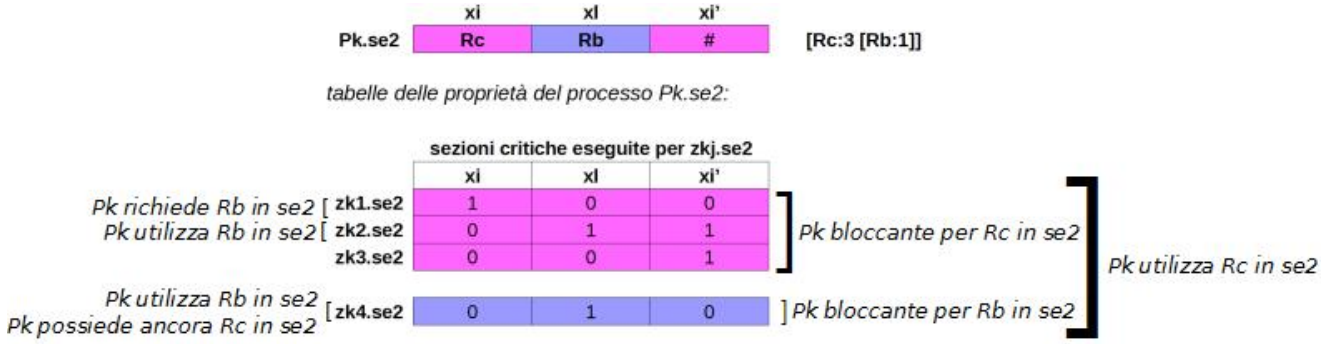
Ora, osservando che *ogni nuovo processo fittizio introdotto* per P_k, presenta in realtà *le medesime caratteristiche di quelli analizzati* nei paragrafi precedenti, conseguentemente è possibile utilizzare riapplicare i ragionamenti di cui prima al fine di ottenere le *tabelle delle proprietà*, *il set di variabili binarie* ed *i vincoli lineari di modello per* $\{P_k.se_1, P_k.se_2, P_k.se_3\}$.

Le figure nella pagina seguente riportano le relazioni esistenti tra:

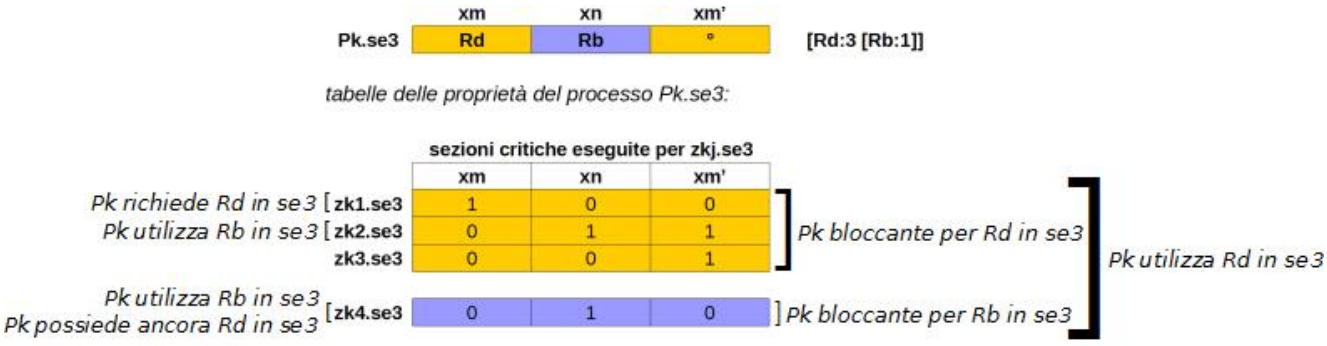
- *combinazioni di esecuzione* $z_{kj}.se_i$ *di* $P_k.se_i \rightarrow$ *sezioni critiche di* $P_k.se_i$ $\{x_a, x_b, x_a'\}$, $\forall i \in \{1, 2, 3\}$ e corrispondentemente $\{x_a, x_b, x_a'\} =_{def} \{x_g, x_h, x_g'\}$, $\{x_i, x_l, x_i'\}$, $\{x_m, x_n, x_m'\}$;
- *le combinazioni di esecuzione* $z_{kj}.se_i$ *di* $P_k.se_i \rightarrow$ *{variabili binarie del modello di* $P_k.se_i$*}*, tali che ogni "*variabile del modello di* $P_k.se_i$ *è a valor logico 1"* in corrispondenza della/dell'*attivazione della combinazioni* $z_{kj}.se_i$ *disposta/e di fianco* in figura.



(a)



(b)



(c)

Figura 2.19 Tabelle per la modellazione ILP dei task $\{P_k.se_{1\,(a)}, P_k.se_{2\,(b)}, P_k.se_{3\,(c)}\}$

2.13 Variabili logiche globali del modello ILP di P_k

Affinché *le interazioni bloccanti tra P_k e gli altri processi del task set* possano essere modellate *senza considerare le singole sezioni critiche esterne dello stesso P_k*, allora per esso è introdotto un set di variabili binarie *globali* che permetta la rappresentazione della proposizione:

- *"P_k possiede la generica proprietà$_x$ rintracciabile in una o più delle sue sezioni critiche*

esterne"

Per il caso del processo P_k in esame, il corrispondente insieme di variabili *globali* è mostrato nella tabella che segue.

Variabili binarie globali di modello per Pk
Ra_MANAGED_BY_Pk *Rb_MANAGED_BY_Pk* *Rc_MANAGED_BY_Pk* *Rd_MANAGED_BY_Pk*
Ra_REQUEST_MATCHED_BY_Pk *Rb_REQUEST_MATCHED_BY_Pk* *Rc_REQUEST_MATCHED_BY_Pk* *Rd_REQUEST_MATCHED_BY_Pk*
Rd_REQUEST_LAUNCHED_BY_Pk

Figura 2.20 Variabili logiche *globali* del modello ILP di P_k

Per rendere più facile la comprensione dell'utilità di tali variabili, si consideri lo scenario in cui P_k fosse *bloccante per il possesso del lock del semaforo di* R_a. In questo caso risulterebbero *attive* le variabili di modello di $P_k.se_1$:

- $\{z_{k1}.se_1$, R_a MANAGED BY $P_k.se_1$, R_a REQUEST MATCHED BY $P_k.se_1$, R_d REQUEST LAUNCHED- BY $P_k.se\}$.

Il comportamento del task in questione sarebbe *simulato attraverso* quindi il processo fittizio $P_k.se_1$, e per il processo P_k risulterebbero pertanto a valore "1" le variabili binarie *globali*:

- $\{R_a$ *MANAGED BY* P_k *,* R_a *REQUEST MATCHED BY* P_k *,* R_d *REQUEST LAUNCHED BY--*$P_k\}$.

2.14 Vincoli per la definizione delle variabili globali di P_k

In virtù dell'esempio appena analizzato, ciascuna delle precedenti *variabili globali* di P_k è "controllata" per mezzo di una *coppia di vincoli*, cioè:

a) *un'equazione lineare che abbia al primo membro la variabile globale di* P_k *che si intende definire*, quindi *al secondo la sommatoria* di quelle *corrispondenti introdotte per i nuovi processi* $\{P_k.se_1, P_k.se_2, P_k.se_3\}$. Seguendo questo criterio, nel modello di P_k compaiono i vincoli per $j \in \{a, b, c, d\}$ (vedi Resource Set $=_{def} \{R_a, R_b, R_c, R_d\}$):

a.1) R_j MANAGED BY $P_k = \Sigma_{i=\{1,2,3\}}$ R_j MANAGED BY $P_k.se_i$;

a.2) R_j REQUEST MATCHED BY $P_k = \Sigma_{i=\{1,2,3\}}$ R_j REQUEST MATCHED BY $P_k.se_i$;

a.3) R_j REQUEST LAUNCHED BY $P_k = \Sigma_{i=\{1,2,3\}}$ R_j REQUEST LAUNCHED BY $P_k.se_i$;

b) *una relazione di mutua esclusione per ciascun tipo di variabile* definita al livello di sezioni critiche esterne se_1, se_2, se_3 di P_k, e cioè:

b.1) $\Sigma_{i=\{1,2,3\}}$ R_j MANAGED BY $P_k.se_i \leq 1$;

b.2) $\Sigma_{i=\{1,2,3\}}$ R_j REQUEST MATCHED BY $P_k.se_i \leq 1$;

b.3) $\Sigma_{i=\{1,2,3\}}$ R_j REQUEST LAUNCHED BY $P_k.se_i \leq 1$.

2.15 Il comportamento di P_k è simulato da uno solo dei processi fittizi $P_k.se_i$

Come si è detto in precedenza, per processi che presentano una struttura analoga a quella di P_k, l'unico *scenario ammissibile* è quello per cui *essi risultino bloccanti attraverso solamente una delle proprie sezioni critiche esterne* (o una in esse annidate).

Ciò comporta la necessità di introdurre per il modello ILP di P_k di una variabile binaria, indicata col nome IS ACTIVE z_{kj} OF $P_k.se_i$, per ogni sezione critica esterna di P_k (in questo caso $\forall i \in \{1,2,3\}$), tale che:

- ciascuna variabile IS ACTIVE z_{kj} OF $P_k.se_i$ assuma valore unitario esclusivamente in corrispondenza della *"selezione"* in soluzione ottima di una delle combinazioni di esecuzione z_{kj} associate alla se_i di P_k.

Il controllo di tali variabili avviene conseguentemente per mezzo dei vincoli lineari *del tipo*:

- IS ACTIVE z_{kj} OF $P_k.se_i = z_{k1}.se_1 \vee z_{k2}.se_i \vee z_{k3}.se_i \vee z_{k4}.se_i$, $\forall i \in \{1,2,3\}$;

ed il relativo significato è: "la variabile IS ACTIVE z_{kj} OF $P_k.se_i$ assume valore unitario qualora P_k sia supposto bloccante per mezzo di una delle proprie combinazioni d'esecuzione $z_{kj}.se_i$ associate alla i-esima sezione critica esterna se_i".

Infine, la seguente relazione di mutua esclusione:

- $\Sigma_{i\in\{1,2,3\}}$ IS ACTIVE z_{kj} OF $P_k.se_i \leq 1$;

consente di rispettare le indicazioni descritte come *scenario ammissibile* all'inizio del paragrafo.

Il modello ILP completo di questo task è disponibile nel Capito V del lavoro di Tesi, con il nome "Modello ILP Applicazione r-t con esempio $P_k =_{def} P_3$ – Cap. 2".

CAPITOLO III

Modellazione delle interazioni bloccanti e del task P_1 di un'applicazione real-time

3.1 Introduzione alla gestione dei blocchi diretti

Le tecniche descritte nel precedente capitolo, utili a rappresentare i processi di un dato Task Set in un modello ILP, costituiscono un primo step *fondamentale* anche per un'opportuna simulazione della presenza di blocchi diretti tra i processi in esame.

L'obiettivo di questo capitolo è dunque quello di spiegare come le variabili di processo considerate in precedenza:

- {R_k REQUEST LAUNCHED BY P_i, R_k REQUEST MATCHED BY P_j};

insieme a quelle (il cui significato sarà chiarito successivamente):

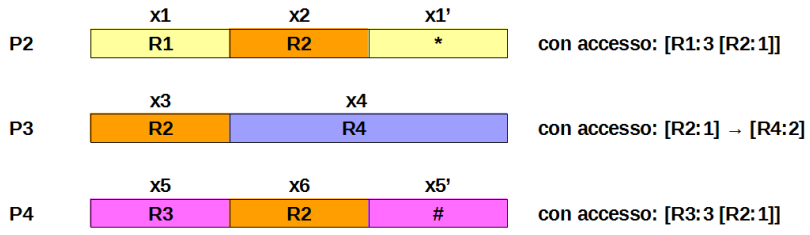
- {DB ON R_k FOR P_i; NO R_k DB FOUND FOR P_i; R_k RELEASED BY P_j FOR P_i};

consentano di tener conto dell'eventuale esistenza di conflitti nell'accesso a risorse condivise, sempre nel contesto della determinazione del massimo tempo di blocco B_{P1} del processo a massima priorità P_1.

3.2 Analisi del Task Set I di riferimento

Per rendere più immediata la spiegazione delle scelte progettuali fatte, si prende ad esempio il caso del Task Set I costituito da tre processi {P_2, P_3, P_4}, con livelli di priorità $p(P_2) > p(P_3) > p(P_4)$, i quali richiedono l'accesso alle risorse {R_1, R_2, R_3, R_4}.

La modalità di accesso agli elementi del Resource Set da parte dei task, e le tabelle che mostrano tutte le loro peculiarità sono mostrate nelle figure 3.1 e 3.2 di sotto.



notazione:

R_1, * ={*esecuzione di istruzioni con lock del semaforo di* R_1};

R_2 ={*esecuzione di istruzioni con lock del semaforo di* R_2};

R_3, # ={*esecuzione di istruzioni con lock del semaforo di* R_3};

R_4 ={*esecuzione di istruzioni con lock del semaforo di* R_4};

Figura 3.1 Task Set I e schema di accesso alle risorse dei processi {P_2, P_3, P_4}

per il processo P2:

	sezioni critiche incluse per zh			zh eseguibile in richiesta di Rj da parte di Pk		Rj richiesta per zh		Rj già in possesso per zh	
	x1	x2	x1'	R1	R2	R1	R2	R1	R2
z21	1	0	0	1	0	0	1	1	0
z22	0	1	1	1	0	0	0	1	1
z23	0	0	1	1	0	0	0	1	0
z24	0	1	0	0	1	0	0	1	1

per il processo P3:

	sezioni critiche incluse per zh		zh eseguibile in richiesta di Rj da parte di Pk		Rj richiesta per zh		Rj già in possesso per zh	
	x3	x4	R2	R4	R2	R4	R2	R4
z31	1	0	1	0	0	0	1	0
z32	0	1	0	1	0	0	0	1

per il processo P4:

	sezioni critiche incluse per zh			zh eseguibile in richiesta di Rj da parte di Pk		Rj richiesta per zh		Rj già in possesso per zh	
	x5	x6	x5'	R3	R2	R1	R2	R1	R2
z41	1	0	0	1	0	0	1	1	0
z42	0	1	1	1	0	0	0	1	1
z43	0	0	1	1	0	0	0	1	0
z44	0	1	0	0	1	0	0	1	1

Figura 3.2 Tabelle delle proprietà dei processi del Task Set I

La Figura 3.2 ricorda come la modalità con cui si assume eseguito un dato processo P_i possa essere rappresentata attraverso una combinazione di esecuzione z_{ih} delle sue sezioni critiche x_k, nonché come ciascuna z_{ih} descriva i seguenti sottoinsiemi per gli elementi del Resource Set:

- RS $\text{Subset}_{1,Pi}$: il sottoinsieme delle risorse per le quali P_i possiede già il lock di semaforo;
- RS $\text{Subset}_{2,Pi}$: il sottoinsieme delle risorse per le quali invece lo stesso processo non possiede il controllo.

3.3 Condizione I necessaria ma non sufficiente per l'esistenza di blocchi diretti

È facile intuire come, ogni qualvolta si assuma che P_i sia supposto eseguito secondo una determinata z_{ih} cui corrisponde un preciso RS $\text{Subset}_{2,Pi}$ *non vuoto* allora, per ogni R_k appartenente a tale sottoinsieme, è necessario che nel modello sia "attivata" la variabile binaria di processo R_k REQUEST LAUNCHED BY P_i.

Questo ragionamento fa capire come la prima condizione necessaria ma *non sufficiente* affinché un processo P_i nella soluzione ottima fornita *abbia subito un blocco diretto* per la risorsa R_k, è:

- **condizione I**: in soluzione ottima $\exists$"$z_{ih} = 1$", tale che per questa combinazione P_i richiede il possesso di R_k, cioè "R_k REQUEST LAUNCHED BY $P_i = 1$".

Considerando il Task Set I in esame:

a) il processo P_2 richiederà l'accesso alla risorsa R_2, quindi sarà valida la relazione "R_2 REQUEST LAUNCHED BY $P_2 = 1$", se *in soluzione ottima risulterà essere non nulla* la combinazione z_{21}.

   Inoltre, questo ragionamento è valido anche per la combinazione z_{41} del processo P_4, per quanto concerne l'*attivazione* di R_2 REQUEST LAUNCHED BY P_4.

b) il processo P_3 invece non disporrà nel proprio modello ILP di alcuna variabile R_k REQUEST LAUNCHED BY P_3, poiché per esso non sono previsti accessi annidati, e dunque non potrà mai subire blocchi diretti nel corso della propria esecuzione.

La prima implicazione logica che scaturisce dalle precedenti affermazioni è costituita dall'introduzione nel modello ILP di *ciascun processo P_i con accessi annidati*, di vincoli *del tipo*:

- **v3.1:** $z_{ih} - R_k \, REQUEST \, LAUNCHED \, BY \, P_i = 0, \, con \{ z_{ih} \mid P_i \, richiede \, l'accesso \, alla \, risorsa \, R_k. \}$

In seconda analisi, se per un dato processo P_i fosse verificata la condizione I di cui prima, allora per esso sarebbe necessario determinare se sia stato effettivamente oggetto di un blocco diretto, oppure alternativamente abbia trovato la risorsa richiesta *libera*.

Ciò è determinabile se per ogni processo P_i, quindi per ogni risorsa R_k *bloccante in modo diretto* per il task in questione, risultano incluse nel suo modello ILP le seguenti variabili binarie:

- DB ON R_k FOR P_i, rappresentante il caso in cui *il processo P_i abbia effettivamente subito un blocco diretto nell'accesso alla risorsa R_k durante la propria esecuzione come z_{ih};*
- NO R_k DB FOR P_i, che in modo complementare alla variabile appena analizzata, è quella che descrive lo scenario in cui P_i abbia richiesto R_k, ma *non abbia subito alcun blocco diretto su di essa*.

Questi ragionamenti in realtà sono preliminari all'inserimento nel modello ILP, dei vincoli:

- **v3.2:** $DB \, ON \, R_k \, FOR \, P_i + NO \, R_k \, DB \, FOR \, P_i - R_k \, REQUEST \, LAUNCHED \, BY \, P_i = 0 \;\; ;$
- **v3.3:** $\sum_i DB \, ON \, R_k \, FOR \, P_i \leq 1 \; ; \;\; \forall P_i \mid \exists R_k \, REQUEST \, LAUNCHED \, BY \, P_i \;\; .$

L'importanza di questa coppia di relazioni per ogni processo P_i che possa richiedere l'accesso ad una data risorsa R_k, risiede nel fatto che:

1. ammettano la possibilità che esista *un solo* task P_i che possa subire un blocco diretto su R_k;
2. forzano gli altri eventuali processi P_j, richiedenti la medesima risorsa, a considerarla libera qualora esista già un processo P_i per cui sia verificata la condizione "DB ON R_k-

-FOR P_i = 1". Essi sono pertanto funzionali affinché per i processi P_j siano "attivate" invece le proprie variabili NO R_k DB FOR P_i;

3. se per P_i non fosse presente in soluzione ottima nessuna delle combinazioni z_{ih} che prevedano l'attivazione della propria variabile binaria R_k REQUEST LAUNCHED BY P_i, allora conseguentemente tale processo non potrà essere incluso:
   i. né nell'insieme dei "processi che possono subire un blocco diretto su R_k";
   ii. né in quello che include i task che, richiedendo l'accesso a tale risorsa, ritrovano questa stessa come "*già liberata"*;
   iii. ed in conclusione, P_i non potrà pertanto causare alcun blocco transitivo su R_k per il processo P_1 di cui si intende stimare il B_{P1}.

La possibilità quindi che una data risorsa R_k possa essere non occupata da alcun task, o possa essere stata già liberata da un altro processo P_i, costituisce la dimostrazione della *non sufficienza* della condizione I precedente.

3.4 Condizione II necessaria ma non sufficiente per l'esistenza di blocchi diretti

La *seconda condizione necessaria ma non sufficiente* affinché un dato processo P_i possa subire un blocco diretto nell'accesso alla risorsa R_k, è che esista in soluzione ottima una combinazione z_{jh} unitaria associata al processo P_j (con $p(P_1) > p(P_j)$), ed indicante che:

1. P_j possiede già il lock di R_k (in questo caso risulterà a *valor unitario* la propria corrispondente variabile di modello {R_k MANAGED BY P_j});
2. P_j esegue tutte le proprie sezioni critiche necessarie al rilascio di R_k.

La variabile binaria adoperata per modellare nel complesso il comportamento appena descritto per il generico processo P_j è quella R_k *REQUEST MATCHED BY* P_j.

3.5 Condizioni necessarie e sufficienti per l'esistenza di un blocco diretto su R_k

Applicando il seguente ragionamento al Task Set I di riferimento ed allo scenario della determinazione del massimo B_{P1}, allora:

- P_2 causerà un blocco diretto per la risorsa R_1, se di esso sarà inclusa in soluzione ottima una delle combinazioni del set {z_{21}, z_{22}, z_{23}}. Mentre per quanto riguarda R_2, il processo P_2 risulterà bloccante se la z_{24} sarà presente con valore '1' in soluzione finale;
- il discorso al punto precedente è analogo per P_4, relativamente alle risorse R_3 ed R_2, con: {R_3 REQUEST MATCHED BY P_4 = 1 ⇔ "z_k=1" per z_k ∈ {z_{41}, z_{42}, z_{43}}} e {R_2 REQUEST MATCHED BY P_4 = 1 ⇔"z_{44}=1"};

- infine, il processo P_3 avrà la facoltà di *rispondere* alternativamente (le sue sezioni critiche difatti non sono annidate) a "richieste di liberazione" per le risorse R_2 ed R_4 se rispettivamente risulteranno attive in soluzione ottima le sue combinazioni z_{31} o z_{32}.

In conclusione: le condizioni di *"esistenza di una richiesta per una risorsa R_k da parte di un processo P_i"* e di *"liberazione immediata di essa da parte di un altro processo P_j"*, sono *necessarie e sufficienti affinché un blocco diretto su R_k influenzi il calcolo della durata del parametro B_{Pl}.*
Tuttavia, queste stesse non comportano automaticamente che tale blocco sia effettivamente subito dal processo P_i, qualora nel Task Set esistano altri processi P_h che possano ugualmente richiedere R_k.

3.6 Condizioni necessarie e sufficienti per "P_i subisce un blocco diretto per R_k da parte di P_j"

Per rendere più chiari i ragionamenti fatti, si considerino i task dell'applicazione r-t in esame, quindi si supponga che il processo P_1 a più elevata priorità preveda di accedere *in sequenza* alle risorse R_1 ed R_3.
Se oltre a ciò fosse verificata anche la condizione per cui i processi P_2 e P_4 possiedano già rispettivamente il lock dei semafori di R_1 e R_3, è evidente che P_1 subirà un blocco diretto da parte di entrambi i task di cui prima.
Il tutto è rappresentato nel modello ILP attraverso:

- l'attivazione da parte di P_1 delle proprie variabili R_1 REQUEST LAUNCHED BY P_1 e R_3 REQUEST LAUNCHED BY P_1;
- l'attivazione da parte di P_2 della sua variabile R_1 REQUEST MATCHED BY P_2;
- l'attivazione da parte di P_4 della sua variabile R_3 REQUEST MATCHED BY P_4.

Sulla base delle precedenti ipotesi, è possibile rilevare come P_1 sia in realtà l'unico a poter inoltrare una richiesta di liberazione per la risorsa R_1, nonché è di fatti anche *l'unico processo* che può potenzialmente *subirne un blocco diretto*.
In questo particolare scenario, l'attivazione quindi della variabile di richiesta di P_1 per R_1 e quella di liberazione corrispondente di P_2, costituiscono una condizione necessaria e sufficiente perché P_1 sia considerato il processo che difatti subisce il blocco diretto sulla risorsa R_1. Tale discorso è ovviamente estendibile relativamente al blocco che subisce P_1 su R_3 da parte di P_4.

Al contrario, la non sufficienza delle precedenti condizioni emerge tutta notando come, nell'ipotesi che la risorsa R_2 sia già occupata dal processo P_3, e che di nuovo P_2 e P_4 posseggano il controllo di R_1 ed R_3, nel tentativo allora di liberare tali risorse per P_1 esse necessiteranno entrambi l'accesso alla risorsa R_2.

I P_2 e P_4 attiveranno rispettivamente le proprie variabili binarie R_2 REQUEST LAUNCHED BY--P_2 e R_2 REQUEST LAUNCHED BY P_4, mentre per il processo P_3 avrà valore unitario R_2 REQUEST MATCHED BY P_3.

A questo punto, è facile comprendere come:

- il processo P_3 possa bloccare al più uno tra P_2 e P_4, richiedenti il possesso di R_2;
- se subisse il blocco diretto il processo P_2, e dunque P_4 ritrovasse R_2 come *"già liberata*", allora sarebbero *attive* nel modello le variabili {DB ON R_2 FOR P_2, NO R_2--DB FOR P_4};
- se subisse viceversa il blocco diretto il processo P_4, e dunque P_2 ritrovasse R_2 come *"già liberata*", allora sarebbero attive le variabili {DB ON R_2 FOR P_4, NO R_2 DB--FOR P_2}, in modo quindi esattamente speculare rispetto alla situazione descritta al punto precedente.

La situazione appena analizzata, fa emergere chiaramente la necessità:

1. di fissare un criterio di ottimizzazione sulla base del quale poter determinare quale dei due processi {P_2,P_4} subisca il blocco diretto su R_2 (vedi "Ordine degli Accessi alle Risorse" da parte dei task dell'applicazione r-t);
2. di disporre per ciascun processo P_j che possa liberare la propria risorsa R_k per un altro processo P_i, di una variabile binaria di "matching", indicata nel seguito come R_k RELEASED BY P_j FOR P_i.

Per quanto riguarda la questione posta nel punto 1) di sopra, si illustrerà adeguatamente nel Capitolo IV come sia possibile introdurre delle variabili di penalizzazioni in Funzione Obiettivo, tali che l'attivazione di specifiche combinazioni di variabili di tipo {DB ON R_k FOR P_i} e {NO R_k DB FOR P_j} *non possano essere ammesse* (si osservi come questi "campi" binari siano associati a processi differenti).

Ad esempio per il set di processi e risorse in analisi, è sufficiente notare come: essendo richiesta dal processo P_1 prima la risorsa R_1 rispetto a quella R_3, allora se nella soluzione ottima al problema della determinazione del massimo B_{P1} fosse inclusa la presenza di un blocco diretto su

R_2 causato da P_3, questo sarebbe certamente subito da P_2. Nelle ipotesi iniziali si è infatti assunto che esso possieda il controllo di R_1, richiesta come *prima risorsa da P_1*. Sarebbe quindi assolutamente inammissibile che il blocco su R_2 determinato da P_3, sia subito dal processo P_4, ed in funzione del rilascio di R_3 per il processo a maggior priorità P_1.

In merito alla questione descritta al punto 2), l'introduzione ed uso delle variabili binarie R_k RELEASED BY P_j FOR P_i avviene per ogni coppia di processi (P_i, P_j) aventi *le stesse caratteristiche* descritte ad esempio per (P_1,P_2), nonché l'"attivazione" di questa variabile coincide con il risultato di un and logico ($\wedge$) associato al verificarsi contemporaneo delle seguenti condizioni:

- "DB ON R_k FOR P_i = 1";
- "R_k REQUEST LAUNCHED BY P_i = 1";
- "R_k REQUEST MATCHED BY P_j = 1".

Questo ragionamento implica l'introduzione della seguente nuova tipologia di vincolo:

- **v3.4:** $$R_k\_RELEASED\_BY\_P_j\_FOR\_P_i = \wedge \begin{Bmatrix} DB\_ON\_R_k\_FOR\_P_i; \\ R_k\_REQUEST\_LAUNCHED\_BY\_P_i; \\ R_k\_REQUEST\_MATCHED\_BY\_P_j; \end{Bmatrix}$$

Inoltre, il modello ILP del processo P_i sarà ulteriormente corredato di un'equazione del tipo:

- **v3.5:** $$DB\_ON\_R_k\_FOR\_P_i - \sum_j R_k\_RELEASED\_BY\_P_j\_FOR\_P_i = 0; \quad \forall P_j \in \{P_j \text{ può causare un blocco diretto su } R_k \text{ per } P_i\}$$

mentre quello del processo P_j presenterà il vincolo di tipo:

- **v3.6:** $$R_k\_REQUEST\_MET\_BY\_P_j - \sum_h R_k\_RELEASED\_BY\_P_j\_FOR\_P_h \leq 0; \quad \forall P_h \in \{P_h \text{ può subire un blocco diretto su } R_k \text{ da parte di } P_j\}$$

Il significato di ciascun elemento del Set di Vincoli {v3.4, v3.5, v3.6} è il seguente:

a) partendo dall'ultimo elemento di questo insieme, si comprende facilmente come: *se un processo P_j "risponda alla richiesta di liberazione della risorsa R_k"* (quindi valga: "R_k REQUEST MET BY P_j = 1"), allora necessariamente tale processo dovrà rilasciare la risorsa in questione per uno dei task che potenzialmente può inoltrarne una richiesta.
   Ergo: *almeno una delle variabili di P_j di tipo {R_k RELEASED BY P_j FOR_P_i} dovrà essere conseguentemente "attivata"*;
b) il secondo vincolo rappresenta invece la proprietà per cui: *se un processo P_i subisce un blocco diretto su una data risorsa R_k, allora necessariamente dovrà essere attiva almeno una delle variabili che prevedono il rilascio di R_k specificatamente per P_i.*

c) infine, il vincolo v3.4 è di fondamentale importanza poiché unisce assieme le condizioni (a) e (b) di cui sopra, esprimendo che: "*la risorsa R_k risulterà rilasciata da P_j per il processo P_i"* se è P_i che, oltre a lanciare la richiesta per tale risorsa ed a incontrarne la relativa risposta da parte P_j, è anche il task che ha la propria variabile di blocco diretto "attivata" (cioè "DB ON R_k FOR P_i = 1").

In realtà, nel modello ILP dell'applicazione r-t è necessario inserire anche le relazioni lineari definite *per ogni singola risorsa R_k del Resource Set*, definite come:

- **v3.7:** $\forall R_k \in Resource\ Set \Rightarrow \sum_i R_k\_REQUEST\_MET\_BY\_P_i \leq 1 \mid \exists R_k\_REQUEST\_MET\_BY\_P_i\ nel\ modello;$
- **v3.8:** $\forall R_k \in Resource\ Set \Rightarrow \sum_{i,j} R_k\_RELEASED\_BY\_P_j\_FOR\_P_i \leq 1 \Leftrightarrow \exists DB\_ON\_R_k\_FOR\_P_i = 1;$

ed aggiornare quello v3.3 come $v3.3_{upgrade}$:

- **$v3.3_{upgrade}$:** $\forall R_k \in Resource\ Set \Rightarrow \sum_i DB\_ON\_R_k\_FOR\_P_i \leq 1 \Leftrightarrow \exists R_k\_REQUEST\_MET\_BY\_P_i = 1.$

Le motivazioni che portano ad introdurre questi vincoli sono le seguenti:

- il vincolo v3.7, indica che potrà rispondere alla "richiesta di liberazione di una data risorsa R_k", *solamente un processo* del Task Set;
- il vincolo v3.8 modella invece lo scenario per cui risulterà attiva al più una variabile di "rilascio" di R_k (vedi: "R_k RELEASED BY P_j FOR P_i = 1"), e se e solo se si è verificato un blocco diretto su questa stessa (cioè esiste una variabile DB ON R_k FOR P_i che sia a valore unitario);
- il vincolo $v3.3_{upgrade}$ ribadisce nuovamente il fatto che un solo processo possa subire un blocco diretto su una data risorsa R_k, ed aggiunge la condizione che effettivamente esiste un blocco se e solo se, in ottica di massimizzazione di B_{P1}, è stata attivata una combinazione z_{ih} di un processo P_i che *risponde* ad una richiesta di liberazione di R_k.

L'importanza di questa serie di equazioni e disequazioni lineari del modello consiste esattamente nel fatto che: esse *stabiliscono una "corrispondenza 1a1" tra il processo che subisce il blocco diretto su una risorsa ed il task che causandolo la rilascia*.

Se a questo punto si volesse applicare i ragionamenti illustrati al caso del Task Set e del Resource Set di riferimento, allora è possibile effettuare le seguenti osservazioni:

1. il modello del processo P_2, disporrà delle variabili:

1.1. {R_1 REQUEST MET BY P_2, R_2 REQUEST MET BY P_2}, per rappresentare lo scenario in cui questo task risponda ad una richiesta di rilascio rispettivamente delle risorse R_1 e R_2;

1.2. {R_2 REQUEST LAUNCHED BY P_2, DB ON R_2 FOR P_2, NO R_2 DB FOR P_2}, utili per modellare la circostanza in cui P_2 non possieda il controllo di R_2, quindi richiedendone l'eventuale accesso possa potenzialmente incappare in un blocco diretto;

1.3. {R_1 RELEASED BY P_2 FOR P_1, R_2 RELEASED BY P_2 FOR P_4}, fondamentali qualora P_2 rispettivamente fosse assunto come *processo bloccante per P_1 relativamente all'accesso alla risorsa R_1*, e per *P_4 per il possesso di R_2*.

2. il processo P_3:

2.1. sarà provvisto delle variabili binarie {R_2 REQUEST MET BY P_3, R_4 REQUEST MET BY P_3}, per rappresentare il caso in cui tale processo risponda ad una richiesta di rilascio rispettivamente delle risorse R_2 e R_4;

2.2. non possiederà alcuna variabile di tipo {R_x REQUEST LAUNCHED BY P_3} in quanto esso non prevede alcun accesso annidato a nessuna ulteriore risorsa per le sue sezioni critiche x_3 e x_4;

2.3. potendo liberare la risorsa R_2 per i processi P_2 e P_4, nel suo set di variabili binarie saranno conseguentemente incluse anche R_2 RELEASED BY P_3 FOR P_2, e R_2 RELEASED BY P_3 FOR P_4.

3. Infine, per il processo P_4 si possono estendere tutte le considerazioni già espresse per P_2, in questo caso applicandole però agli scenari per cui: R_3 sia richiesta da P_1; e P_4, utilizzando R_2, possa subire blocchi diretti causati da P_2 o P_3, od eventualmente determinarne uno per P_2.

3.7 Modellazione implicita dei blocchi transitivi

Quella descritta nell'attuale Capitolo è una strategia particolarmente *furba* di modellazione dei blocchi *diretti* tra processi di un'applicazione r-t. Tale affermazione è giustificata dal fatto che è sufficiente determinare quali siano le risorse ed i task su cui ciascun processo può vicendevolmente *subire e causare* un blocco diretto, per riuscire a rappresentare in una *modalità implicita* anche i blocchi *transitivi*.

In merito a ciò, un esempio pratico lo si può ricavare dal Task Set di riferimento constatando come, in virtù del fatto che:

(a) il processo a maggior priorità P_1 possa subire un blocco diretto su R_1 e R_3, da parte rispettivamente di P_2 e P_4;

(b) P_2 sia invece potenzialmente soggetto ad un blocco diretto su R_2 da parte di P_3 o P_4;

(c) ed infine, P_4 può essere arrestato una volta ereditata la priorità $p(P_1)$, da P_2 o P_3, anch'esso relativamente al possesso di R_2;

risulta evidente allora che, in ottica di ricerca del massimo tempo di blocco di B_{P1}, sarà modellato in soluzione finale anche il *blocco transitivo di P_1 su R_2 da parte di P_3*, determinato implicitamente:

- attraverso il *blocco diretto di P_1 su R_1 da parte di P_2, e quello di P_2 su R_2 causato da P_3*;
- oppure di per mezzo dei blocchi diretti di *P_1 su R_3 da P_4, e di quest'ultimo per R_2 a causa sempre di P_3.*

L'efficacia delle scelte progettuali discusse si può dimostrare anche decidendo di imporre un *valore negativo* al coefficiente che in Funzione Obiettivo rappresenta la durata temporale della combinazione z_{31} (cioè $c\_z_{31} < 0$), così da indurre l'*assenza di blocchi diretti sulla risorsa R_2 da parte di P_3, né per il processo P_2, né per quello P_4.*

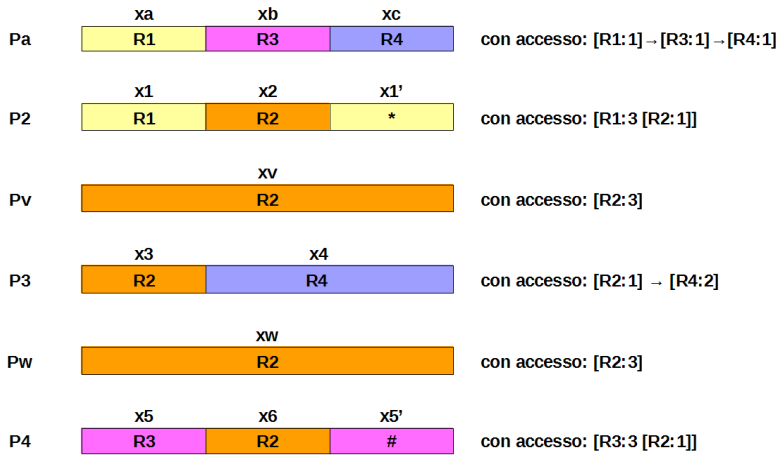
Un'ulteriore osservazione è infine fatta relativamente alla presenza della sezione critica x_4 di P_3, associata all'utilizzo della risorsa R_4. Essa è del tutto irrilevante ai fini del calcolo del tempo di blocco B_{P1} di P_1, poiché nessun processo all'infuori di P_4 richiede l'utilizzo di questa risorsa.
Al contrario, il suo ruolo diverrebbe significativo se ad esempio P_1 richiedesse ulteriormente il lock del semaforo di R_4, dopo aver ottenuto l'accesso a R_1 ed a R_3.
Nell'ipotesi infatti che $c\_z_{31} << c\_z_{32}$ allora, in ottica di massimizzazione del valore di B_{P1}, il blocco diretto di P_1 su R_4 da parti di P_3 risulterebbe incluso nella soluzione finale in luogo di quello transitivo *"P_1 blocco transitivo su (P_3,R_2)"*.
In questo caso la risorsa R_2 sarebbe considerata pertanto libera, ed i processi P_2 e P_4 non subirebbero nessuna interruzione nel corso della loro esecuzione per accedervi.

3.8 Analisi del Task Set II di riferimento

Lo scenario appena descritto si complicherebbe ancor di più se si aggiungesse l'ipotesi che nel Task Set in esame esistano due ulteriori processi P_v e P_w che, descritti rispettivamente dalle sezioni critiche x_v e x_w su R_2, abbiano livelli di priorità p_v e p_w tali che:

- $p(P_a) > p(P_2) > p(P_v) > p(P_3) > p(P_w) > p(P_4)$ (nota: in questa discussione si assume che P_a sia il processo più critico, indicato precedentemente come task P_1).

In queste circostanze diviene necessaria l'introduzione nel modello di una nuova tipologia di vincoli, per evitare che siano fornite alcune soluzioni ottime che si dimostrano essere in realtà *inammissibili* per l'insieme di regole associate alla schedulazione RMPO ed al Protocollo PI.
Lo schema riassuntivo del nuovo Task Set è mostrato nella figura 3.3 seguente.



notazione:

R_1, * ={*esecuzione di istruzioni con lock del semaforo di* R_1};

R_2 ={*esecuzione di istruzioni con lock del semaforo di* R_2};

R_3, # ={*esecuzione di istruzioni con lock del semaforo di* R_3};

R4 ={*esecuzione di istruzioni con lock del semaforo di R4*};

Figura 3.3 Task set II di riferimento

Estensione temporale della sezione xk di Pi		[t.u.]
P2	δ_x1	3
	δ_x2	1
	δ_x1'	1
Pv	δ_xv	3
P3	δ_x3	1
	δ_x4	2
Pw	δ_xw	3
P4	δ_x5	3
	δ_x6	1
	δ_x5'	1

coefficiente in F.O. della combinazione zik di Pi		
P2	c_z21	3
	c_z22	2
	c_z23	1
	c_z24	1
Pv	c_zv1	3
P3	c_z31	1
	c_z32	2
Pw	c_zw1	3
P4	c_z41	3
	c_z42	2
	c_z43	1
	c_z44	1

Figura 3.4 Tabelle delle durate temporali δx_k delle sezioni critiche x_k e tabella dei coefficienti c_z_k associati alle combinazioni d'esecuzione z_k dei processi del Task Set II

Analizzando la durata temporale delle singole sezioni critiche ed i coefficienti in funzione obiettivo per le combinazioni d'esecuzione dei processi del Task Set II, è facile comprendere

come la massimizzazione del tempo di blocco B_{Pa} del processo P_a si ottiene se in soluzione ottima assumono valore unitario le seguenti z_k:

- z_{21} di P_2, che produce un blocco diretto per P_a nell'accesso ad R_1;
- z_{41} di P_4, che produce un blocco diretto per P_a nell'accesso ad R_3;
- z_{32} di P_3, che produce un blocco diretto per P_a nell'accesso ad R_4;
- z_{v1} di P_v, che produce un blocco diretto per P_2 nell'accesso a R_2, quindi *transitivo* per P_a.

Le ragioni che portano a considerare questa soluzione come *ottimizzante della funzione obiettivo del problema ILP formulato*, ed *ammissibile* per quanto concerne *il complesso di regole descritte dalla strategia di schedulazione RMPO e dal protocollo PI*, sono le seguenti:

a) del processo P_3 si sceglie la combinazione di esecuzione con durata maggiore, quindi in questo caso la z_{32}. Ciò esclude ovviamente la possibilità che questo stesso task produca un blocco diretto anche per R_2 (con z_{31}), dato che P_3 utilizza tali risorse in modalità *sequenziale e non annidata, e dunque può bloccare un processo a priorità più elevata al più per il possesso di una sola risorsa.*

b) essendo presente in soluzione ottima la combinazione z_{32}, allora la risorsa R_2 sarà stata già liberata dallo stesso P_3, nonché eventualmente occupata da un processo con priorità superiore a $p(P_3)$. Nel caso non fosse verificata quest'ultima condizione, P_3 avrebbe infatti proseguito la sua esecuzione sino al completamento, e non potrebbe causare quindi alcun blocco diretto su R_4 per P_a.

   In modo più dettagliato, P_3 *riprende la sua esecuzione* solamente in corrispondenza della richiesta da parte di P_a della risorsa R_4. Pertanto: o R_2 è difatti occupata da un processo P_k con $p(P_k) > p(P_3)$ e tale da arrestare P_3 quando esso abbia già iniziato l'esecuzione di x_4 (così da bloccare poi successivamente P_a), oppure R_2 è semplicemente libera.

   Ad esempio, se l'accesso di P_a alle sue risorse di cui necessita fosse nel seguente ordine: $R_3 \rightarrow R_1 \rightarrow R_4$, allora si verificherebbe esattamente il caso per cui un processo P_j con $p(P_j)<p(P_3)$, e cioè P_4, nel tentativo di liberare R_3 accede in modalità annidata a R_2 e, nell'ottica di massimizzazione di B_{Pa}, esso la trova occupa nuovamente da P_v (esattamente secondo la regola di cui sopra), poiché se fosse occupata da P_w allora questo implicherebbe che $p(P_w) > p(P_3)$ per poter arrestare P_3 in possesso di R_4, violando però così le ipotesi iniziali sui livelli di priorità.

Il principio che è possibile derivare è il seguente:

- ogniqualvolta risulti rilasciata una generica risorsa R_k da un processo P_j, questa potrà essere occupata esclusivamente da un processo P_i tale che $p(P_i)>p(P_j)$.

Nel caso di esecuzione della z_{32} da parte di P_3, allora non è possibile che R_2 sia occupata da P_w poiché i livelli priorità sono tali che $p(P_w) < p(P_3)$. È invece ammesso che R_2 sia in possesso di P_v, dato che $p(P_v) > p(P_3)$.

3.9 Gestione del lock di risorse già utilizzate da processi del Task Set

Per determinare l'insieme di vincoli che consente la rappresentazione della proprietà appena discussa nei modelli ILP delle applicazioni r-t, si riprendere in esame il processo P_i del Caso di Studio I del precedente capitolo.

La prima osservazione che è possibile fare a riguardo, è:

- a causa della *struttura con annidamento delle sue sezioni critiche*, P_i potrebbe bloccare un processo P_k a priorità superiore anche qualora:
  - esso abbia già rilasciato R_4, e possieda il lock dei semafori di $\{R_1, R_2, R_3\}$;
  - nell'ipotesi in cui abbia rilasciato le risorse $\{R_3, R_4\}$, ma possieda quelle $\{R_1, R_2\}$;
  - ed infine, qualora non possieda più $\{R_2, R_3, R_4\}$, e disponga solo del lock di R_1.

Per ogni risorsa che possa essere stata rilasciata precedentemente da un generico processo P_i, supposto bloccante attraverso una propria combinazione z_k:

- è necessario introdurre nel modello ILP dell'applicazione un'opportuna variabile binaria, indicata come $\{R_x$ LOCK AVAILABLE FOR P_j WITH pr_j HIGHER THAN $pr_i\}$;

tale che: ogniqualvolta essa assuma valore unitario, allora il lock del semaforo che gestisce la risorsa R_x potrà essere occupato solo e soltanto da un processo P_j con $p(P_j) > p(P_i)$.

Relativamente al caso del processo P_i in esame:

- in corrispondenza della "selezione" in soluzione ottima di una delle proprie combinazioni $\{z_6, z_7, z_{11}, z_{12}, z_{15}\}$, per la risorsa R_4 sarà vincolata ad assumere valore unitario la variabile logica R_4 LOCK AVAILABLE FOR P_j WITH pr_j HIGHER THAN pr_i;
- in corrispondenza della "selezione" in soluzione ottima di una delle proprie combinazioni $\{z_6, z_7, z_{12}\}$, per la risorsa R_3 sarà vincolata ad assumere valore unitario la variabile logica R_3 LOCK AVAILABLE FOR P_j WITH pr_j HIGHER THAN pr_i;
- in corrispondenza della "selezione" in soluzione ottima della propria combinazione $\{z_7\}$, allora per la risorsa R_2 risulterà ad "1" la variabile binaria R_2 LOCK AVAILABLE FOR--P_j WITH pr_j HIGHER THAN pr_i.

La semplice selezione in soluzione ottima delle z_k indicate negli insiemi nell'elenco precedente costituiscono in realtà una *condizione necessaria ma non sufficiente* affinché possano essere vincolate ad assumere valore unitario le variabili "di tipo" R_x LOCK AVAILABLE FOR P_j--WITH pr_j HIGHER THAN pr_i.

Ad esempio: affinché le risorse $\{R_2, R_3, R_4\}$ possano essere considerate come "già rilasciate" da parte di P_i attraverso selezione di z_7, richiede in realtà "che siano a valor nullo anche tutte le restanti combinazioni z_k di P_i".

Si noti infatti che la condizione "P_i ha rilasciato anticipatamente R_x", si verifica in corrispondenza:

1. della selezione in soluzione ottima di combinazioni descritte da soli *residui* di sezioni critiche di processo (vedi anche *sezioni critiche pendenti*)*;*
2. ed ogniqualvolta ciascuna z_j che compare nei nodi *di livello superiore* del diagramma ad albero relativo alla Catena di Attivazione che descrive gli scenari "compatibili con l'utilizzo di z_k", risulti essere nulla.

Un esempio a tal riguardo potrebbe essere rappresentato dal caso in cui $\{R_3,R_4\}$ siano supposte già rilasciate da P_i a causa della selezione di z_{12} in soluzione ottima. Se si analizzasse il diagramma completo delle catene di attivazione di questo processo, si potrebbe facilmente notare come affinché possa considerarsi verificata la condizione "$\{R_3, R_4\}$ già utilizzate e liberate da P_i":

- è necessario che siano allora nulle tutte le restanti z_j di P_i, meno la z_7, poiché essa *compare come unico nodo figlio dell'albero che include* z_{12}.

Estendendo i ragionamenti appena fatti a ciascuna combinazione "che possa rappresentare difatti un rilascio precedentemente avvenuto di risorse da parte di P_i", allora diventa evidente la necessità d'introdurre una corrispondente variabile binaria, indicata come AND HIGHERprPROCS FOR z_j, e ognuna di esse sarà descritta da un and logico:

- **v3.9:**

$$\forall z_k \equiv \sum x_j',\ \textit{cioè descritta dall'esecuzione di residui}\ x_j' \Rightarrow \exists\ \textit{AND HIGHERprPROCS FOR}\ z_k$$
$$\textit{AND HIGNERprPROCS FOR}\ z_k = z_k \wedge \{ \wedge_j \overline{z_j} \}$$
$$\forall z_j \notin \{\textit{set dei nodi figli della catena di attivazione con nodo radice in}\ z_k\}$$

Inoltre, ogni variabile di tipo R_x LOCK AVAILABLE FOR P_j WITH pr_j HIGHER THAN pr_i risulterà associata ad un or logico definito come:

- **v3.10:**

$$\forall R_x \mid R_x\_MANAGED\_BY\_P_i = 0 \text{ per } z_k \text{ di } P_i \text{ e } z_k = \sum x_j' \Rightarrow$$
$$\Rightarrow \exists R_x \, LOCK\ AVAILABLE\ FOR\ P_j\ WITH\ pr_j\ HIGHER\ THAN\ pr_i$$
$$R_x \, LOCK\ AVAILABLE\ FOR\ P_j\ WITH\ pr_j\ HIGHER\ THAN\ pr_i = \vee_k \{AND\ HIGHERprPROCS\ FOR\ z_k\};$$

L'*attivazione* di una variabile {R_x LOCK AVAILABLE FOR P_j WITH pr_j HIGHER THAN pr_i}, implicando che il possesso della risorsa R_x possa essere esclusivamente associato ad un processo P_k che abbia priorità $p(P_k)>p(P_i)$, allora rende necessaria l'introduzione di un ulteriore vincolo *di mutua esclusione* nel modello ILP, come:

- **v3.11:**

$$R_x \, LOCK\ AVAILABLE\ FOR\ P_j\ WITH\ pr_j\ HIGHER\ THAN\ pr_i + \sum R_x\_MANAGED\_BY\_P_k \leq 1$$
$$\forall P_k \mid pr_k \leq pr_i.$$

L'ultimo vincolo v3.11 introdotto offre lo spunto per una fondamentale precisazione relativamente all'utilizzo delle variabili binarie R_x *MANAGED BY* P_i, necessarie per la *corretta determinazione* del tempo di blocco B_{P1}.

Come già detto nel capitolo precedente, ogniqualvolta risulti selezionata in soluzione ottima una combinazione z_k di P_i, allora conseguentemente risulteranno *"attivate"* ciascuna delle variabili R_x MANAGED BY P_i predisposte ad indicare di quali risorse R_x il processo P_i possieda già il lock.

Le relazioni che permettono di vincolare a valor unitario tali variabili sono presenti negli specifici modelli di processo, e dei semplici esempi dei vincoli di attivazione di queste variabili sono presenti nel precedente capitolo per i Casi di Studio II e III.

L'importanza attribuibile a tali variabili consiste nel fatto che:

- adoperando esse in opportuni *vincoli di mutua esclusione*, allora è possibile rappresentare nel modello la regola per cui: "il *lock bloccante* di ogni risorsa R_x, può essere attribuito ad uno solo dei processi P_i con priorità inferiore a quella di P_1".

Ciò si traduce nel modello ILP, nell'utilizzo di vincoli di tipo:

- **v3.12:** $\sum R_x\_MANAGED\_BY\_P_k \leq 1; \forall P_k \in \{P_k \text{ utilizza } R_x\} \text{ e } \forall R_x \in Resource\ Set.$

È evidente pertanto che la relazione v3.11 impone un'ulteriore restrizione relativamente al possesso delle risorse del Resource Set, rispetto a quello v3.12 appena descritto.

3.10 Analisi del Task Set III e nuove considerazioni per quello in Figura 2.9 - Capitolo II

Si supponga di prendere in esame l'applicazione mostrata nella Figura 3.5 di sotto, indicata come Task Set III, e di voler ulteriormente analizzare anche quella rappresentata in Figura 2.9 del precedente Capitolo II.

I modelli ILP utili per la determinazione del massimo tempo di blocco B_{P1} del processo a maggior priorità P_1 per entrambi le applicazioni considerate sono rilevanti in quanto in ciascuno di essi si fa uso di ognuna delle nuove tipologie di vincoli finora discusse.

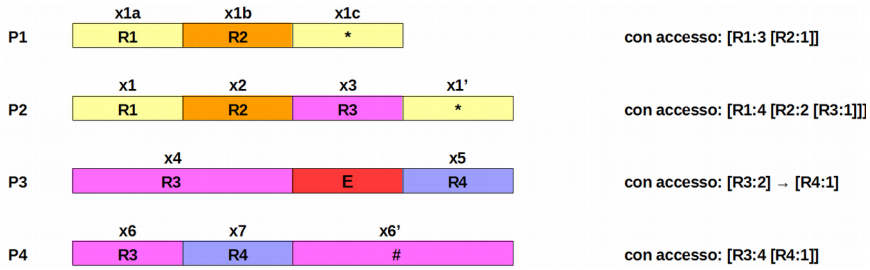


Figura 3.5 Task Set III di riferimento

Estensione temporale della sezione xk di Pi		[t.u.]
P2	δ_x1	3
	δ_x2	2
	δ_x3	1
	δ_x1'	1
P3	δ_x4	2
	δ_x5	1
P4	δ_x6	4
	δ_x7	1
	δ_x6'	2

coefficiente in F.O. della combinazione zik di Pi		
P2	c_z21	4
	c_z22	3
	c_z23	2
	c_z24	1
	c_z25	2
	c_z26a	1
	c_z26b	1
P3	c_z31	2
	c_z32	1
P4	c_z41	4
	c_z42	3
	c_z43	2
	c_z44	1

Figura 3.6 Tabelle delle durate temporali δx_k delle sezioni critiche x_k e tabella dei coefficienti c_z_k associati alle combinazioni d'esecuzione z_k dei processi del Task Set III

Esaminando la soluzione del problema descritto dal modello ILP formulato (entrambi disponibili nel Capitolo V come "Modello ILP Applicazione r-t Fig. 3.5 – Cap. 3"), è evidente come:

1. il tempo di blocco B_{P1} stimato per il processo P_1 sia pari a 8 t.u.;
2. il blocco subito da P_1 è costituito da:
   - 2.1. un blocco diretto subito da P_1 nell'accesso alla risorsa R_1, causato dal task P_2;
   - 2.2. un blocco diretto subito da P_2 alla risorsa R_3, ipotizzata detenuta da P_4.

Il processo P_1 quindi subisce un blocco transitivo associato alla risorsa R_3, e non ne subisce alcuno nel corso del suo accesso alla risorsa R_2.

Conseguentemente la liberazione di R_1 da parte di P_2 per P_1, ed il rilascio di R_3 da parte di P_4 per P_2, determina un'interruzione di P_1 per 8 t.u. definita dalla *selezione* in soluzione ottima delle combinazioni z_{21} di P_2 e z_{41} di P_4.

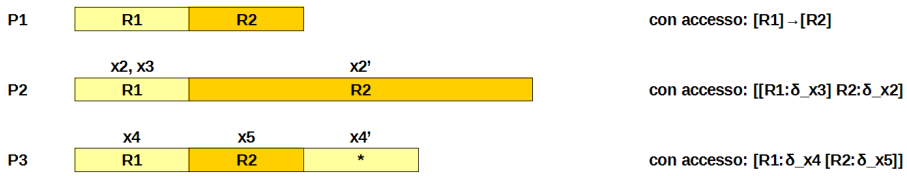


Figura 3.7 Riproposizione Task Set di fig. 2.9 – Cap. II

Riprendendo ora il caso di studio già discusso nel Capitolo II, ed il cui diagramma è nuovamente riportato nella figura 3.7, appare in primis necessario determinare il modello del processo P_2, dato che la struttura d'accesso alle risorse condivise di questo task evidenzia singolarità non individuate analizzando la struttura dei processi degli altri casi di studio finora considerati.
Le tabelle che riassumono le proprietà principali di questo task ed il relativo Modello ILP sono riportati di seguito:

	sezioni critiche incluse per zh			zh eseguibile in richiesta di Rj		Rj richiesta per zh		Rj già in possesso per zh		sezioni pendenti per zh	
	x2	x3	x2'	R1	R2	R1	R2	R1	R2	R1	R2
z21	1	0	0	0	1	0	0	1	1	°	°
z22	0	0	1	0	1	0	0	0	1	°	°
z23	0	1	0	1	0	0	0	1	1	°	x2', z22

Figura 3.8 Tabelle delle principali proprietà del processo P_2

v1: z21 + z22 + z23 - P2_WILL_USE_R2_BY_z22 <= 1

v2: z21 + z22 <= 1

v3: z23 - P2_WILL_USE_R2_BY_z22 = 0
v4: z21 + P2_WILL_USE_R2_BY_z22 <= 1

v5: z21 + z23 <= 1

v6: R1_REQUEST_MATCHED_BY_P2 - z23 = 0
v7: R2_REQUEST_MATCHED_BY_P2 - z21 - z22 = 0

v8: R1_MANAGED_BY_P2 - z21 - z23 = 0

v9: R2_MANAGED_BY_P2 - z21 >= 0
v10: R2_MANAGED_BY_P2 - z22 >= 0
v11: R2_MANAGED_BY_P2 - z23 >= 0
v12: R2_MANAGED_BY_P2 - z21 - z22 - z23 <= 0

v13: AND_HIGHERprPROCS_FOR_z22 - z22 <= 0
v14: AND_HIGHERprPROCS_FOR_z22 + z21 <= 1
v15: AND_HIGHERprPROCS_FOR_z22 + z23 <= 1
v16: AND_HIGHERprPROCS_FOR_z22 + z21 + z23 - z22 >= 0

v17: R1_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr2 - AND_HIGHERprPROCS_FOR_z22 = 0

Figura 3.9 Vincoli che modellano P_2

La definizione del set di equazioni e disequazioni lineari che rappresentano il task P_2 avviene sulla base degli stessi criteri esposti nel Capitolo II, nonché le osservazioni di maggior importanza sono:

- il modello del task P_2 non include nessuna variabile di tipo R_k REQUEST LAUNCHED BY P_2, e conseguentemente di tipo DB ON R_k FOR P_2 e NO DB ON R_k BY P_2 con $k \in \{1,2\}$, poiché data la struttura degli accessi alle risorse R_1 ed R_2, il task in questione non inoltrerà mai alcuna richiesta di liberazione relativamente ad esse;
- le combinazioni di esecuzione z_{21} e z_{23} rappresentano rispettivamente l'esecuzione integrale per P_2 delle sezioni critiche x_2 e x_3. Quindi per z_{21} è previsto il rilascio da parte di P_2 di $\{R_1, R_2\}$, mentre per la seconda della sola R_1;
- la combinazione z_{23} può essere "selezionata" in soluzione ottima assieme a quella z_{22}, così da simulare per P_2 la possibilità che esso rilascio in modo *discontinuo nel tempo* le risorse R_1 e R_2 qualora ne possedesse il lock per entrambe (vedi anche variabile di P_2WILL--USE R_2 BY z_{22});
- i vincoli di "tipo HRprPROCs" per il possesso della risorsa R_1 si dimostrano fondamentali poiché, per la tecnica ideata per simulare i blocchi diretti tra processi, il task P_3 potrebbe subirne uno su R_2 da parte di P_2.

In relazione a quanto affermato nell'ultimo punto di sopra, ed analizzando la modalità con cui P_2 utilizza le risorse $\{R_1,R_2\}$, è possibile osservare come: se in soluzione ottima comparisse la configurazione $\{z_{21}, z_{23}=0; z_{22}=1\}$ per P_2, allora necessariamente: la risorsa R_1 potrà essere occupata solo e soltanto da un eventuale processo P_i con priorità $p(P_i)$ tale che "$p(P_1) > p(P_i) > p(P_2)$".

Pertanto, l'introduzione dei vincoli di "tipo HRprProcs" nel modello di P_2 è necessaria in questo scenario:

1. per favorire l'occupazione della risorsa R_1 da parte di un eventuale task P_i che *abbia un livello di priorità superiore a quello del processo P_2*;
2. per evitare che il possesso della risorsa R_1 sia attribuita erroneamente ad un processo P_j con priorità $p(P_j)<p(P_2)$.

Per il caso di studio in esame, se non fossero introdotte le equazioni e le disequazioni lineari di *tipo HRprProcs* per il modello P_2, allora in ottica di massimizzazione della funzione obiettivo del problema ILP per la determinazione del parametro B_{P1} del processo P_1, sarebbe consentita la

“selezione” in soluzione ottima delle seguenti combinazioni di esecuzione (e che in realtà non potrebbero mai comparire assieme con valore unitario):

a) z_{31}, che modella un blocco diretto di P_1 causato dal processo P_3, supposto quest'ultimo in possesso di R_1;
b) z_{22}, che implicherebbe invece un blocco diretto di P_3 nell'accesso ad R_2 e causato da P_2.

L'introduzione dei vincoli di tipo HRprProcs nel modello del task P_2 sono pertanto fondamentali per escludere la possibilità che sia fornita come ottima la soluzione *inammissibile* appena descritta.
Con semplici ragionamenti, è altresì possibile comprendere come per questa applicazione gli unici scenari ammissibili siano:

1. nel caso in cui valga la relazione "$\delta\_x_4 < (\delta\_x_3 + \delta\_x_2')$", il processo P_1 subisce esattamente due blocchi diretti associati al possesso delle risorse R_1 e R_2, entrambi causati da P_2. In soluzione ottima ciò si traduce con la presenza delle sole combinazioni z_{22} e z_{23} a valor unitario.
2. se invece risulta verificata "$\delta\_x_4 > (\delta\_x_3 + \delta\_x_2')$", il processo P_1 subisce un blocco diretto solamente sulla risorsa R_1, causato da P_3, e trova invece libera la risorsa R_2 (su di essa non subirà nessun blocco nemmeno il processo P_3 durante l'esecuzione della sua sezione critica più esterna).

Gli scenari appena analizzati sono stati simulati opportunamente adoperando il Modello ILP presente nel Capitolo V di questo lavoro di Tesi con il nome “Modello ILP Applicazione r-t Fig. 3.7 – Cap. 3” (nota: in esso sono definite due opportune *funzioni obiettivo* che permettono di verificare la validità di quanto appena affermato. Inoltre, i risultati corrispondenti sono mostrati nei in corrispondenza dei paragrafi “Soluzione Test I” e “Soluzione Test II”).
In conclusione, è evidente come il primo dei contesti d’esecuzione considerati mostri l'importanza della rappresentazione delle sezioni critiche pendenti per i modelli dei processi di un dato task set.

3.11 Variabili binarie e relazioni lineari del modello ILP del processo P_1

Nel primo capitolo di questo elaborato si è discusso dell’insieme di vincoli che consentono di rappresentare dettagliatamente le proprietà dei task che concorrono alla definizione dei blocchi diretti e/o transitivi per il processo P_1 (a massima priorità), di cui *si intende calcolare il*

valore del massimo tempo di blocco B_{Pl}. Un problema sinora non affrontato è quello relativo alla *modellazione del processo P_l.*

In primo luogo è possibile affermare che la rappresentazione di P_1, all'interno del modello ILP dell'applicazione r-t in esame, avviene adoperando un *sottoinsieme di equazioni e disequazioni lineari* strettamente incluso in quello delle relazioni utili a modellare lo stesso task qualora fosse tra i *processi a priorità inferiore potenzialmente bloccanti*.

Questa affermazione è giustificata dal fatto che per il processo P_1:

1. non è necessario introdurre nel modello finale per il calcolo di B_{P1} le relative combinazioni di esecuzione z_{1k} ed i vincoli di gestione associati, in quanto nessuna di queste variabili binarie comparirà in funzione obiettivo per definire un *contributo bloccante*;
2. è irrilevante per esso simulare la liberazione di risorse *di cui possiede il lock*, in quanto P_1 è il processo a maggior priorità, pertanto *non potrà mai causare il blocco diretto per un altro processo*;
3. è inutile determinare le equazioni e le disequazioni che permettano la modellazione del *rilascio discontinuo di risorse* da parte di P_1, nonché descrittive del comportamento di questo processo relativamente alla gestione delle sezioni critiche pendenti che ad esso possono essere attribuite.

Al contrario, per P_1 è di fondamentale importanza introdurre nel modello ILP dell'applicazione r-t, l'insieme di vincoli con cui sono simulati i meccanismi per mezzo dei quali questo task *richiede ed ottiene il controllo del lock dei semafori delle risorse di cui necessita*.
La *filosofia* che spiega le scelte progettuali fatte è la seguente:

- per determinare il tempo di blocco B_{P1} del processo P_1, si ipotizza che esso sia lanciato in esecuzione senza che possegga alcuno dei lock dei semafori che gestiscono le risorse di cui necessita nel corso della propria esecuzione. Conseguentemente *per ciascuna di esse*, tale processo *inoltrerà una richiesta di liberazione*, incontrando o meno un blocco a seconda che quest'ultimo contribuisca a massimizzare il parametro B_{P1}.

Per rendere maggiormente comprensibile quanto descritto, si prende in esame un semplice esempio in cui il processo P_1 di cui si intende determinare il massimo B_{P1}, è costituito:

a) da una prima sezione critica x_1 che prevede l'accesso ed utilizzo della risorsa R_1;
b) da un'ulteriore sezione critica x_2, associata all'impiego da parte di questo processo di R_3.

Lo schema sintetico della struttura del processo in analisi è mostrata nella figura 3.10 seguente.

Figura 3.10 Struttura del processo P_1

In funzione delle ipotesi fatte, i vincoli che nel modello ILP finale saranno introdotti per il processo P_1 riguarderanno:

1. l'inoltro da parte di questo task della richiesta di liberazione della risorsa R_1, con potenziale blocco diretto su di essa;
2. il lancio di un'ulteriore richiesta di rilascio (ancora una volta potenzialmente bloccante), in questo caso per la risorsa R_3.

Facendo riferimento ora alle discussioni affrontate nei paragrafi precedenti, le *tipologie di vincoli lineari* necessari per rappresentare P_1 sono quelle indicate come $\{v_{3.2}; v_{3.4}; v_{3.5}\}$, per $R_y \in \{R_1, R_3\}$. Essi sono riportati sinteticamente di sotto.

$$\textit{vincoli per la gestione del blocco diretto del processo } P_1 \textit{ su } R_y,\ \forall R_y \in \{R_1, R_3\}:$$

$$v_{1.P1.Ry}: DB\,ON\,R_y\,FOR\,P_1 + NO\,R_y\,DB\,FOR\,P_1 = R_y\,REQUEST\,LAUNCHED\,BY\,P_1;$$

$$v_{2.P1.Ry}: R_y\,RELEASED\,BY\,P_j\,FOR\,P_1 = DB\,ON\,R_y\,FOR\,P_1 \wedge R_y\,REQUEST\,LAUNCHED\,BY\,P_1 \wedge R_y\,REQUEST\,MATCHED\,BY\,P_j$$

$$v_{3.P1.Ry}: DB\,ON\,R_y\,FOR\,P_1 = \sum_j R_y\,RELEASED\,BY\,P_j\,FOR\,P_1$$

$$v_{a.Pj.Ry}: R_y\,REQUEST\,MATCHED\,BY\,P_j \le \sum_k R_y\,RELEASED\,BY\,P_j\,FOR\,P_k + R_y\,RELEASED\,BY\,P_j\,FOR\,P_1$$
$$con: P_j \in \{\textit{processi che causano un blocco diretto su } R_y \textit{ per } P_1\}$$

Un'importante osservazione circa il set di relazioni lineari appena mostrato, riguarda i *processi* P_j e le relative variabili binarie che di essi compaiono. In particolare, essendo questi i *possibili processi che causano un blocco diretto su* R_y *per* P_1, ne consegue che:

a) le variabili R_y RELEASED BY P_j FOR P_1 che compaiono nel vincolo $v_3.P_1.R_y$, sono quelle che stabiliscono la presenza di un blocco diretto di P_1 su R_y (vedi DB ON R_y FOR P_1);
b) la variabile attraverso la quale *si modella il rilascio da parte di* P_j *della risorsa* R_y *per* P_1, indicata come $\{R_y$ RELEASED BY P_j FOR $P_1\}$, è presente nella sommatoria al secondo

membro del vincolo $v_a.P_j.R_y$, cioè tra quelle *attivabili* corrispondentemente da $\{R_y \text{ REQUEST MATCHED BY } P_j\}$.

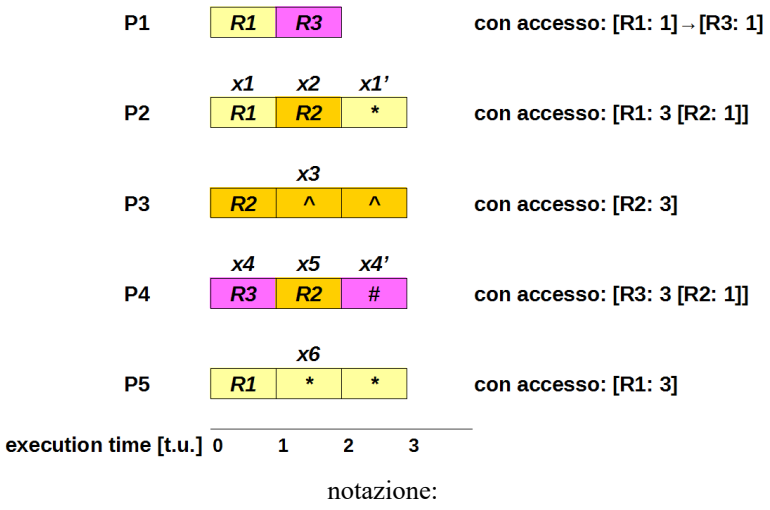
CAPITOLO IV

Relazioni per l'ordinamento temporale nell'accesso alle risorse

4.1 Soluzioni ottime non congruenti con l'ordine degli accessi alle risorse

I modelli ILP sinora costruiti per la determinazione del tempo di blocco B_{P1} del processo P_1, possono fornire in realtà soluzioni ottime potenzialmente *incongruenti* con l'ordine con cui i task accedono alle risorse.

Un esempio di quanto affermato si può ottenere esaminando l'applicazione r-t mostrata di sotto:



notazione:

R_1, *={*esecuzione di istruzioni con lock del semaforo di R_1*}

R_2, ^={*esecuzione di istruzioni con lock del semaforo di R_2*}

R_3, #={*esecuzione di istruzioni con lock del semaforo di R_3*}

Figura 4.1 Task set di riferimento per l'analisi dell'ordine degli accessi

Analizzando lo schema degli accessi dei processi in questione, e notando che gli unici a possedere accessi annidati sono $\{P_2,P_4\}$, nonché ricercando l'insieme di sezioni critiche bloccanti per P_1 che ne massimizzano il corrispondente parametro B_{P1}, allora *la soluzione ottima* è descritta da:

- P_1 subisce un blocco diretto per il lock del semaforo che gestisce la risorsa R_1, supposto detenuto da P_2;
- P_1 subisce un blocco diretto in corrispondenza dell'accesso alla risorsa R_3, per la quale il lock del proprio semaforo è posseduto da P_4;
- P_1 soffre di un *blocco transitivo* su R_2, determinato da P_3.

Le regole di modellazione sinora utilizzate non vietano però che in soluzione finale il blocco transitivo subito da P_1 per R_2 sia associato allo scenario: *"P_4 richiedendo l'accesso alla risorsa R_2, subisce un blocco diretto da parte di P_3"*, implicando inoltre che il processo P_2 non abbia subito alcun arresto della propria esecuzione per l'accesso a tale risorsa, e la abbia ritrovata altresì *già libera* durante la propria esecuzione.
Quanto appena descritto, costituisce una soluzione ottima *non congruente con l'ordine di accesso delle risorse bloccanti*, poiché: se ci fosse un blocco transitivo per P_1 su R_2, esso dovrebbe necessariamente essere associato all'accesso annidato di P_2 su R_2, e ciò è evidentemente giustificato dal fatto che *questo task è bloccante per P_1 temporalmente prima rispetto a P_4*.
Pertanto, nel caso in cui P_2 fornisse un contributo non nullo alla massimizzazione del parametro B_{P1}, allora esso sarebbe *l'unico passibile di un blocco diretto su R_2*.

4.2 Albero degli Accessi

Quanto descritto nel precedente paragrafo fa emergere la necessità di definire un ulteriore insieme di regole con le quali sia possibile introdurre un nuovo set di vincoli lineari del modello ILP dell'applicazione r-t in esame, che consentano di tener conto *anche dell'ordine secondo il quale ciascun processo accede alle risorse* di cui necessita nel corso della propria esecuzione.
Le idee di base adoperate in questo ambito sono:

i. costruire un *"Albero degli Accessi"* per ogni processo P_i incluso nel Task Set, e tale che: *l'Albero degli Accessi di P_i è per definizione un set di variabili binarie e vincoli lineari che consentano di tener conto se il processo in questione, o un task che ne abbia causato l'arresto, abbia richiesto ed eventualmente subito un blocco diretto per la risorsa R_h,(per ogni R_h appartenente al Resource Set dell'applicazione r-t).*
ii. adoperando queste strutture informative e le variabili di modello che rappresentano le possibili interazioni bloccanti tra i task, è garantita la *realizzazione semplice e immediata* di un "*Albero Globale degli Accessi" associato nel suo complesso all'applicazione r-t considerata*, e che vincoli il software di ottimizzazione a fornire soluzioni ottime che non violino l'ordine di accesso alle risorse dei processi.

Riprendendo il caso di studio del paragrafo 4.1, l'Albero degli Accessi relativo al processo P_1 è mostrato nell'immagine seguente.

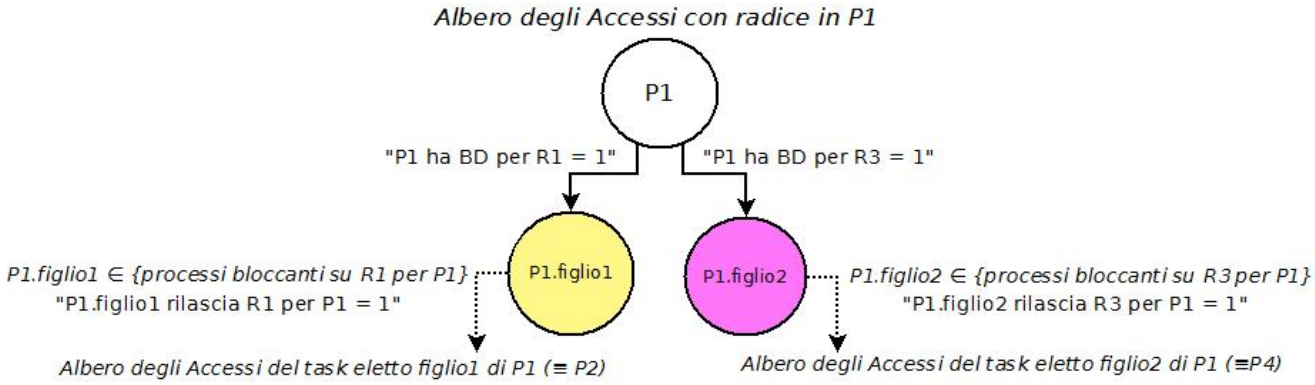


Figura 4.2 Albero degli Accessi per il processo P_1

Osservando la struttura in questione, è facile comprendere come, nell'ipotesi di disporre:

i. dell'*Albero* A_1 del task bloccante su R_1 per P_1 (indicato come: *"Processo Figlio*$_1$ *di* P_1*"*);

ii. dell'*Albero* A_2 del task bloccante su R_3 per P_1 (indicato come: *"Processo Figlio*$_2$ *di* P_1*"*);

e disponendo per P_1, quindi per ogni R_h inclusa nel Resource Set, di una coppia di variabili binarie {x(proposizione$_1$); x(proposizione$_2$)} funzioni di:

- proposizione$_1$: {*in* A_1 è richiesta R_h *e non si verifica* alcun blocco diretto};
- proposizione$_2$: {*in* A_2 è richiesta R_h *e si verifica* un blocco diretto};

conseguentemente *per forzare* la scelta del software di ottimizzazione a fornire soluzioni congruenti con l'ordine con cui possono essere utilizzate le risorse, è sufficiente inserire in funzione obiettivo un'ulteriore variabile logica, *con valore grandemente negativo* ed indicata come *"sequenza non ammessa per {*R_h*:ALBERO* P_1*}"*, tale che:

- sequenza non ammessa per {R_h:ALBERO P_1} $=_{def}$ {x(proposizione$_1$) $\wedge$ x(proposizione$_2$)}.

In questo modo infatti è possibile *penalizzare* la scelta di soluzioni ottime *"inammissibili"* come quella descritta nel paragrafo 4.1.

L'obiettivo dei successivi paragrafi è quindi quello di illustrare *dettagliatamente* quali sono i ragionamenti con cui poter costruire l'*albero degli Accessi di ciascun processo*, e quello *globale associato all'applicazione r-t in esame*, nonché poter disporre di un corrispondente insieme di vincoli lineari nel modello ILP utili per lo scopo appena mostrato in questo paragrafo.

4.3 Albero di P_i: principio I d'implementazione e definizione di $Figlio_j$ di P_i

Il primo principio su cui si incentra l'implementazione delle strutture informative in discussione, per un generico processo P_i è:

- dato P_i incluso nel Task Set di riferimento, per ogni nuova risorsa R_x per cui P_i *inoltri una richiesta di liberazione del lock del relativo semaforo*, esiste potenzialmente *un nodo figlio* nell'Albero degli Accessi *con radice in* P_i.

Figura 4.3 Schema degli accessi alle risorse per il processo P_1

Relativamente al caso del processo P_1, il cui *schema degli accessi* è riportato nella figura 4.3, è facile osservare come:

1. esso richieda in primo luogo l'accesso alla risorsa R_1. Ciò implica che: se su tale risorsa P_1 subisse un *blocco diretto* da parte di un processo P_k con $p(P_k) < p(P_1)$, allora P_k sarebbe considerato il *Processo Figlio n°1* di P_1 (vedi *"SON_1 di P_1"*). Viceversa: *se P_1 non subisse alcun blocco diretto nell'accesso ad R_1*, allora conseguentemente *non esisterebbe alcun Figlio I* per questo processo;
2. dopo aver utilizzato R_1, P_1 richiede l'utilizzo della risorsa R_3. Analogamente a quanto descritto nel punto nella pagina precedente, se l'esecuzione di P_1 fosse interrotta per un blocco diretto determinato dal processo P_j a priorità inferiore, allora quest'ultimo sarebbe considerato come il *Figlio n°2 di P_1* (vedi *"SON_2 di P_1"*). Al contrario se P_1 non fosse interrotto nel corso dell'operazione di accesso ad R_3, allora il suo *albero degli accessi non disporrebbe di alcun Figlio II*.

È evidente che quando si parla di *un Processo Figlio i-esimo nell'Albero degli Accessi di un generico task P_i*, si faccia riferimento ad un processo che *abbia causato un blocco diretto per P_i*. Ritornando dunque nel merito della costruzione dell'*Albero di P_1*, i task *eleggibili* a *$Figlio_1$* e *$Figlio_2$* sono:

- *Processo $Figlio_1$ (vedi SON_1) $\in \{P_2, P_5\}$*, e tale che: *Processo $Figlio_1 \equiv P_2 \Leftrightarrow$ è verificata la proposizione "P_2 ha rilasciato R_1 per P_1"*, o alternativamente *Processo $Figlio_1 \equiv P_5 \Leftrightarrow$ è vera "P_5 rilascia R_1 per P_1"*;
- *Processo $Figlio_2$ (vedi SON_2) $\in \{P_4\}$*, e tale che: *Processo $Figlio_2 \equiv P_4 \Leftrightarrow$ è verificata la proposizione "P_4 rilascia R_3 per P_1"*.

L'utilizzo delle proposizioni *"P_k rilascia R_x per P_i" per identificare il Processo Figlio di P_i nell'l'accesso alla risorsa R_x*, è giustificato dal fatto che i modelli ILP descritti nei capitoli precedenti dispongano già di variabili binarie associate a queste proposizioni (vedi *variabili binarie R_x RELEASED BY P_k FOR P_i*), e cioè che consentano di identificare univocamente quale processo P_k del Task Set abbia bloccato direttamente P_i per il possesso del lock del semaforo di R_x.

4.4 Albero di P_i: principio II d'implementazione, condizione di non congruenza per specifiche soluzioni ottime e relativo meccanismo di penalizzazione

Individuato *il set {Figlio$_1$ di P_i, Figlio$_2$ di P_i, … , Figlio$_n$ di P_i} del task P_i*, il secondo principio per la costruzione delle strutture informative che tengano conto dell'ordine di utilizzo delle risorse stabilisce che *sia necessario, per ciascun Processo Figlio$_j$ di P_i*, quindi *per ogni elemento R_h del Resource Set*, *conoscere*:

a) *se l'Albero degli Accessi del Figlio$_j$ di P_i presenti una richiesta di liberazione del lock del semaforo di R_h*;
b) *se l'Albero degli Accessi del Figlio$_j$ di P_i presenti un blocco diretto per la risorsa R_h*.

Quanto appena detto si traduce nell'introduzione per il modello ILP di P_i di *una coppia di variabili binarie* indicate come {DB ON R_h UNDER SON_k OF P_i; REQUEST FOR R_h UNDER--SON_k OF P_i} per ogni *processo Figlio$_k$ di P_i*, e per ogni elemento del Resource Set, e tali che siano rispettivamente *funzioni delle proposizioni* indicate ai punti a) e b) di sopra.

Il verificarsi pertanto per i Figli g-esimo e i-esimo, con $i > g$, delle condizioni:

1. "REQUEST FOR R_h UNDER SON_g OF $P_i = 1$" $\wedge$ "DB ON R_h UNDER SON_g OF $P_i = 0$";
2. "REQUEST FOR R_h UNDER SON_i OF $P_i = 1$" $\wedge$ ""DB ON R_h UNDER SON_i OF $P_i = 1$";

rappresenta uno *scenario inammissibile* poiché se i Figli g-esimo ed i-esimo richiedessero entrambi l'utilizzo della risorsa R_h, tenendo conto che il *Figlio$_g$ è bloccante temporalmente prima rispetto al Figlio$_i$*, allora:

- il primo di questi task *è l'unico tra i due* a poter subire un blocco diretto su R_h;
- mentre il *Processo Figlio$_i$ ritroverà necessariamente R_h già libera* in corrispondenza del proprio accesso.

Quanto discusso è di fondamentale importanza per *deprecare la selezione di soluzioni ottime non conformi con l'ordine di accesso delle risorse*, in quanto consente l'utilizzo in funzione obiettivo

di opportune variabili logiche per il modello di P_i che abbiano coefficiente grandemente negativo e definite come (per ogni R_h appartenente al Resource Set):

- P_i MODEL BAD SEQUENCE FOR $R_h =_{def}$ {

  $\vee$ [condizione 1) per SON_1 $\wedge$ condizione 2) per SON_2];

  $\vee$ [condizione 1) per SON_1 $\wedge$ condizione 2) per SON_3];

  $\vee$ [condizione 1) per SON_g $\wedge$ condizione 2) per SON_j];

  … }

  per ogni coppia {SON_g; SON_j} di P_i tali che j > g;

e che *penalizzano* l'eventuale scelta di soluzioni ottime *che soddisfano queste condizioni non ammesse*.

4.5 Albero Globale degli Accessi come interconnessione di quelli dei singoli task P_i

Per rendere la *costruzione* dell'Albero di P_i *indipendente* dalla conoscenza specifica dei nodi che compaiono nelle analoghe strutture informative previste per i task P_k *suoi potenziali Processi Figli*, allora *ciascun Albero degli Accessi di un generico processo* P_k è provvisto, per ogni elemento R_h del Resource Set, di una coppia di variabili binarie:

- {REQUEST FOR R_h WHEN P_k IS FATHER; DB ON R_h WHEN P_k IS FATHER}.

Il loro scopo è quello di tener traccia degli *eventi relativi alla risorsa* R_h, scatenati da P_k o dai processi che appaiono come *nodi figli a qualsiasi livello* nell'Albero di P_k.

Grazie a queste variabili si crea *un'interfaccia tra il processo* P_i, assunto come *nodo padre*, e *l'intera struttura ad albero con radice in* P_k, qualora lo stesso P_k fosse *il Processo Figlio j-esimo di* P_i.

Quanto descritto consente inoltre una gestione semplice di scenari del tipo:

i. "per la risorsa R_h, un processo P_k può essere bloccato direttamente da N possibili Processi Figli {P_g, P_h, P_i, …}";
ii. ciascun Processo Figlio di cui sopra, dispone *iterativamente* di una *propria* struttura ad albero, nonché è ammesso che esso sia *il potenziale figlio* di M task dell'applicazione.

In questa situazione, la chiave per realizzazione *un'operazione di join* tra *Albero del Processo Padre* P_k ed *Albero di* P_m *eletto a Processo Figlio*$_i$ è costituita dal fatto quest'ultimo rilascia la risorsa bloccante R_h specificatamente per P_k.

In conclusione, la costruzione dell'Albero Globale associato all'intera applicazione r-t avviene sulla base di semplici interconnessioni tra gli *Alberi degli Accessi dei singoli Processi,* ciascuno

costituito da due soli livelli, cioè: quello del Nodo Padre (Livello I), e quello descritto dai Nodi Figli (Livello II).

Nei prossimi paragrafi è mostrato un *utilizzo pratico* dei precedenti ragionamenti teorici, supposto che questi siano applicati al caso di studio mostrato in figura 4.1.

4.6 Implementazione dell'Albero degli Accessi del task P_1

	Variabili binarie per l'Albero degli Accessi di P1
Variabili d'interfaccia	*REQUEST_FOR_R1_WHEN_P1_IS_FATHER* *REQUEST_FOR_R2_WHEN_P1_IS_FATHER* *REQUEST_FOR_R3_WHEN_P1_IS_FATHER* *DB_ON_R1_WHEN_P1_IS_FATHER* *DB_ON_R2_WHEN_P1_IS_FATHER* *DB_ON_R3_WHEN_P1_IS_FATHER*
Variabili per il SON1 di P1	*NO_SON1_FOR_P1* *REQUEST_FOR_R1_UNDER_SON1_OF_P1* *REQUEST_FOR_R2_UNDER_SON1_OF_P1* *REQUEST_FOR_R3_UNDER_SON1_OF_P1* *DB_ON_R1_UNDER_SON1_OF_P1* *DB_ON_R2_UNDER_SON1_OF_P1* *DB_ON_R3_UNDER_SON1_OF_P1*
Variabili per il SON2 di P1	*NO_SON2_FOR_P1* *REQUEST_FOR_R1_UNDER_SON2_OF_P1* *REQUEST_FOR_R2_UNDER_SON2_OF_P1* *REQUEST_FOR_R3_UNDER_SON2_OF_P1* *DB_ON_R1_UNDER_SON2_OF_P1* *DB_ON_R2_UNDER_SON2_OF_P1* *DB_ON_R3_UNDER_SON2_OF_P1*

Figura 4.4 Variabili di modello per l'implementazione dell'Albero degli Accessi del task P_1

Considerando il processo P_1, la figura di sopra mostra l'elenco delle variabili binarie adoperate per la rappresentazione del relativo Albero degli Accessi nel Modello ILP dell'applicazione r-t di riferimento.

In primo luogo, la presenza di variabili associate *al SON_1 di P_1* è giustificata dal fatto che questo processo inoltri *una prima richiesta* per l'acquisizione della risorsa R_1. La *seconda richiesta di P_1, definita per R_3*, motiva invece l'introduzione delle variabili *per il SON_2*.

Le variabili $\{NO\ SON_i\ FOR\ P_1\}$ per $i \in \{1,2\}$, hanno il compito di indicare se P_1 possieda il proprio Figlio i-esimo (SON_i). Ne consegue che:

1. siccome: "$\exists$ $Figlio_i$ di P_1 $\Leftrightarrow$ P_1 subisce un blocco diretto per la i-esima risorsa da esso richiesta", allora nel modello ILP saranno necessariamente presenti i vincoli lineari:
   - "$NO\ SON_i\ FOR\ P_1 + DB\ ON\ R_j\ FOR\ P_1 = 1$", con i = {1,2} rispettivamente per j = {1,3};

rendendo così *vere* le implicazioni:

- "DB ON R_j FOR $P_1 = 0 \Rightarrow$ NO SON_i FOR $P_1 = 1$", con i = {1,2} rispettivamente per j = {1,3};

2 se valesse "NO SON_i FOR $P_1 = 1$", allora P_1 non ha il $Figlio_i$, pertanto tutte le variabili che offrono informazioni sull'Albero con radice in SON_i, debbono essere vincolate ad assumere valor nullo. I corrispondenti vincoli sono quindi:

- "NO SON_i FOR P_1 + variabile SON_i di $P_1 \leq 1$", con i = {1,2} e $\forall$variabile associata al SON_i.

Anche se P_1 è il processo a più elevata priorità, quindi *non è bloccante per nessun task* dell'applicazione considerata, ne potrà conseguentemente mai comparire come *nodo figlio* di un Albero degli Accessi di un processo incluso nel Task Set di riferimento, a puro titolo esemplificativo sono illustrati nel paragrafo seguente i vincoli di definizione delle *variabili di interfaccia dell'Albero di P_1.*

Per P_1, i vincoli introdotti nel modello ILP per R_j = {R_1, R_3} sono:

a) REQUEST FOR R_j WHEN P_1 IS FATHER $=_{def}$ {
   $\vee$ R_j REQUEST LAUNCHED BY P_1 ;
   $\vee$ R_j REQUEST UNDER SON_1 OF P_1 ;
   $\vee$ R_j REQUEST UNDER SON_2 OF P_1
   };

b) DB ON R_j WHEN P_1 IS FATHER $=_{def}$ {
   $\vee$ DB ON R_j FOR P_1 ;
   $\vee$ DB ON R_j UNDER SON_1 P_1 ;
   $\vee$ DB ON R_j UNDER SON_2 P_1
   };

e per ogni $R_k \in$ Resource Set \ {R_1,R_3}:

c) REQUEST FOR R_k WHEN P_1 IS FATHER $=_{def}$ {
   $\vee$ R_k REQUEST UNDER SON_1 OF P_1;
   $\vee$ R_k REQUEST UNDER SON_2 OF P_1
   };

d) DB ON R_k WHEN P_1 IS FATHER $=_{def}$ {
   $\vee$ DB ON R_k UNDER SON_1 OF P_1 ;
   $\vee$ DB ON R_k UNDER SON_2 OF P_1
   }.

Le relazioni lineari ai punti a) e b) indicano rispettivamente che *nell'Albero degli Accessi con nodo radice in P_1 "esiste una richiesta d'accesso/un blocco diretto per la risorsa R_j" qualora*: "essa/o sia inoltrata/fosse subito dal processo P_1", oppure se "essa/o è generata/subito da uno dei processi inclusi nell'Albero degli Accessi del Processo $Figlio_1$ (vedi SON_1), o del Processo $Figlio_2$ (cioè SON_2)".
I vincoli ai punti c) e d) si differenziano da quelli appena descritti semplicemente per il fatto che P_1 non prevede richieste d'accesso, ne quindi potrà subire blocchi diretti, per le R_k diverse da $\{R_1; R_3\}$.
Pertanto, nella definizione di queste relazioni non compaiono *le corrispondenti variabili di P_1*, ma solo quelle che rappresentano tali proprietà a livello di Albero degli Accessi con radice nei nodi figli SON_1 e SON_2.

L'ultimo step per la realizzazione della struttura ad albero di P_1 riguarda la definizione delle relazioni esistenti tra le *variabili dei SON_1 e SON_2 di P_1* e le effettive strutture albero disposte per i task *che possono ricoprire i ruoli di Processo $Figlio_1$ e Processo $Figlio_2$ di P_1.*
Ricordando infatti che:

a) per P_1, il Processo Figlio1 debba necessariamente appartenere all'insieme $\{P_2,P_5\}$, mentre quello $Figlio_2$ di P_1 può essere rappresentato al più da P_4;
b) ciascun task P_k dell'applicazione dispone di un proprio Albero degli Accessi e di *una coppia di variabili* $\{\text{REQUEST FOR } R_j \text{ WHEN } P_k \text{ IS FATHER; DB ON } R_j \text{ WHEN } P_k \text{ IS FATHER}\}$ $\forall R_j \in \text{Resource Set}$;

allora le relazioni di modello che vincolano i valori binari assunti dalle variabili del Processo $Figlio_1$ (vedi SON_1) e del processo Figlio2 (vedi SON_2) dell'Albero di P_1 sono:

1. $\text{REQUEST FOR } R_j \text{ UNDER SON1 OF } P_1 =_{def} \{$
   $\vee [R_1 \text{ RELEASED BY } P_2 \text{ FOR } P_1 \wedge \text{REQUEST FOR } R_j \text{ WHEN } P_2 \text{ IS FATHER}];$
   $\vee [R_1 \text{ RELEASED BY } P_5 \text{ FOR } P_1 \wedge \text{REQUEST FOR Rj WHEN } P_5 \text{ IS FATHER}];$
   $\};$
2. $\text{DB ON } R_j \text{ UNDER } SON_1 \text{ OF } P_1 =_{def} \{$
   $\vee [R_1 \text{ RELEASED BY } P_2 \text{ FOR } P_1 \wedge \text{DB ON } R_j \text{ WHEN } P_2 \text{ IS FATHER}];$
   $\vee [R_1 \text{ RELEASED BY } P_5 \text{ FOR } P_1 \wedge \text{DB ON } R_j \text{ WHEN } P_5 \text{ IS FATHER}];$
   $\}$
3. $\text{REQUEST FOR } R_j \text{ UNDER } SON_2 \text{ OF } P_1 =_{def}$
   $[R_3 \text{ RELEASED BY } P_4 \text{ FOR } P_1 \wedge \text{REQUEST FOR } R_j \text{ WHEN } P_4 \text{ IS FATHER}];$

4. DB ON R_j UNDER SON_2 OF P_1 $=_{def}$

   $[R_3$ RELEASED BY P_4 FOR $P_1 \wedge$ DB ON R_j WHEN P_4 IS FATHER$]$.

Il significato dei vincoli indicati ai punti 1) e 2) è:

- nell'albero *con radice in Processo Figlio$_1$ di P_1* si verifica "il lancio di una richiesta"/"un blocco diretto per R_j"se: P_2 ricopre il *ruolo di primo figlio*, bloccando direttamente P_1 su R_1 (vedi utilizzo variabile {R_1 RELEASED BY P_2 FOR P_1} del modello ILP di P_2) e se nell'Albero degli Accessi di P_2 esiste una "richiesta inoltrata"/"un blocco diretto per R_j" (vedi: utilizzo variabili di interfaccia {REQUEST FOR R_j WHEN P_2 IS FATHER, DB ON R_j WHEN P_2 IS FATHER} dell'Albero di P_2). Questo discorso deve essere esteso ovviamente anche al task P_5 (ciò giustifica la definizione delle *variabili del SON$_1$* attraverso relazioni di Or Logico).

Considerando invece le relazioni 3) e 4) associate al controllo delle variabili del Processo Figlio$_2$ di P_1:

- anche in questa situazione "si verifica il lancio di una richiesta"/"un blocco diretto per R_j" nell'Albero degli Accessi con radice in SON_2 se e solo se: "P_4 blocca direttamente P_1 su R_3", e quindi ne è il Figlio$_2$, e se nell'Albero degli Accessi dello stesso P_4 "esiste una richiesta inoltrata"/"si verifica un blocco diretto per R_j" (una piccola nota: in questo contesto i vincoli lineari in esame non sono definiti utilizzando alcun Or Logico, in quanto si ricorda che *P_4 è l'unico* task a poter essere *eletto Processo Figlio$_2$ di P_1*).

4.7 Implementazione degli Alberi degli Accessi di {P_2,P_4} e {P_3,P_5}

	Variabili binarie per l'Albero degli Accessi di P2
Variabili d'interfaccia di P2	*REQUEST_FOR_R1_WHEN_P2_IS_FATHER* *REQUEST_FOR_R2_WHEN_P2_IS_FATHER* *REQUEST_FOR_R3_WHEN_P2_IS_FATHER* *DB_ON_R1_WHEN_P2_IS_FATHER* *DB_ON_R2_WHEN_P2_IS_FATHER* *DB_ON_R3_WHEN_P2_IS_FATHER*
Variabili per il SON1 di P2	*NO_SON1_FOR_P2* *REQUEST_FOR_R1_UNDER_SON1_OF_P2* *REQUEST_FOR_R2_UNDER_SON1_OF_P2* *REQUEST_FOR_R3_UNDER_SON1_OF_P2* *DB_ON_R1_UNDER_SON1_OF_P2* *DB_ON_R2_UNDER_SON1_OF_P2* *DB_ON_R3_UNDER_SON1_OF_P2*

	Variabili binarie per l'Albero degli Accessi di P4
Variabili d'interfaccia di P4	*REQUEST_FOR_R1_WHEN_P4_IS_FATHER* *REQUEST_FOR_R2_WHEN_P4_IS_FATHER* *REQUEST_FOR_R3_WHEN_P4_IS_FATHER* *DB_ON_R1_WHEN_P4_IS_FATHER* *DB_ON_R2_WHEN_P4_IS_FATHER* *DB_ON_R3_WHEN_P4_IS_FATHER*
Variabili per il SON1 di P4	*NO_SON1_FOR_P4* *REQUEST_FOR_R1_UNDER_SON1_OF_P4* *REQUEST_FOR_R2_UNDER_SON1_OF_P4* *REQUEST_FOR_R3_UNDER_SON1_OF_P4* *DB_ON_R1_UNDER_SON1_OF_P4* *DB_ON_R2_UNDER_SON1_OF_P4* *DB_ON_R3_UNDER_SON1_OF_P4*

Figura 4.5 Variabili di modello per l'implementazione degli Albero dei task {P_2,P_4}

Per proseguire l'implementazione della struttura informativa *globale* che indichi nel complesso tutti *i possibili ordini di accesso associati all'applicazione r-t in esame*, nella figura precedente è mostrato il set di variabili utili alla realizzazione degli Alberi di P_2 e di P_4.

Osservando *lo schema di accesso alle risorse di tali processi*, è possibile notare come entrambi prevedano *l'utilizzo in modalità annidata di una sola risorsa*, in particolare quella R_2. Estendendo pertanto a questi task i ragionamenti descritti per P_1, è evidente come *P_2 e P_4 possano disporre potenzialmente di un Processo Figlio nella propria struttura ad albero*.

Dovendo ora individuare per essi, l'insieme di di task che possono determinare un loro blocco diretto su R_2, allora:

1. risultano eleggibili per il ruolo di Processo $Figlio_1$ di P_2 quelli $\{P_3,P_4\}$;
2. mentre i task che possono ricoprire tale ruolo per P_4 sono $\{P_2,P_3\}$.

Tutto ciò può essere rappresentato sinteticamente sfruttando *le proprietà dell'Albero degli Accessi di P_2 e P_4,* ed *alcune specifiche variabili di modello ILP dei processi potenzialmente bloccanti per essi*. E cioè:

- "il Processo $Figlio_1$ di P_2 è P_3 $\Leftrightarrow$$P_3$ rilascia R_2 per P_2";
- "il Processo $Figlio_1$ di P_2 è P_4 $\Leftrightarrow$$P_4$ rilascia R_2 per P_2";
- "il Processo $Figlio_1$ di P_4 è P_2 $\Leftrightarrow$$P_2$ rilascia R_2 per P_4";
- "il Processo $Figlio_1$ di P_4 è P_3 $\Leftrightarrow$$P_3$ rilascia R_2 per P_4".

In conclusione, i vincoli che *controllano* i valori assunti:

a) dalle variabili di interfaccia degli Alberi di P_2 e P_4;

b) dalle variabili del "SON_1" delle strutture informative in questione con radice in $\{P_2,P_4\}$;

sono realizzati nelle stesse modalità illustrate nel paragrafo precedente per P_1, e cioè adoperando *ricorsivamente* le variabili definite "a livello di SON_1", quindi quelle che descrivono le proprietà circa le interazioni con i processi *potenzialmente bloccanti* elencati in precedenza, e quelle di interfaccia definite per gli Alberi degli Accessi di $\{P_2, P_3, P_4\}$.

	Variabili binarie per l'Albero degli Accessi di P3
Variabili d'interfaccia di P3	*REQUEST_FOR_R1_WHEN_P3_IS_FATHER* *REQUEST_FOR_R2_WHEN_P3_IS_FATHER* *REQUEST_FOR_R3_WHEN_P3_IS_FATHER* *DB_ON_R1_WHEN_P3_IS_FATHER* *DB_ON_R2_WHEN_P3_IS_FATHER* *DB_ON_R3_WHEN_P3_IS_FATHER*

	Variabili binarie per l'Albero degli Accessi di P5
Variabili d'interfaccia di P5	*REQUEST_FOR_R1_WHEN_P5_IS_FATHER* *REQUEST_FOR_R2_WHEN_P5_IS_FATHER* *REQUEST_FOR_R3_WHEN_P5_IS_FATHER* *DB_ON_R1_WHEN_P5_IS_FATHER* *DB_ON_R2_WHEN_P5_IS_FATHER* *DB_ON_R3_WHEN_P5_IS_FATHER*

Figura 4.6 Variabili di modello per l'implementazione degli Albero dei task $\{P_3,P_5\}$

Nella Figura 4.6 appena mostrata, sono riportate le variabili utili per la costruzione degli Alberi degli Accessi dei processi P_3 e P_5. Quest'ultimo passaggio è infatti fondamentale per garantire una realizzazione corretta dell'*Albero Globale* associato all'applicazione r-t in esame.

A differenza dei task precedentemente analizzati, P_3 e P_5 non prevedono accessi annidati ad elementi presenti nel Resource Set. Pertanto, non potendo *richiedere nel corso della propria esecuzione l'utilizzo di una nuova risorsa,* conseguentemente *non presenteranno mai alcun Processo Figlio* nella propria struttura ad albero.

La prima conclusione logica pertanto è: per rappresentare gli Alberi di P_3 e P_5 sono utilizzate nel modello ILP solamente le relative *variabili di interfaccia.*

La seconda deduzione riguarda il fatto che questi task potranno *ricoprire esclusivamente il ruolo di Processi Figli* per gli Alberi dei restanti processi dell'applicazione, quindi compariranno al più come *Nodi Foglie dell'Albero Globale.* Questo implica che ciascuna variabile di cui prima, debba essere vincolata ad assumere valore nullo per rappresentare che: *"al di sotto dei Nodi di P_3/P_5" non potrà mai verificarsi alcun evento circa la gestione di ogni R_h appartenente al Resource Set.*

In conclusione, i vincoli di modello degli Alberi di $\{P_3, P_5\}$ sono:

- REQUEST FOR R_h WHEN P_k IS FATHER = 0;
- DB ON R_h WHEN P_k IS FATHER = 0;

  $\forall P_k \in \{P_3, P_5\}$, $\forall R_h \in$ Resource Set.

4.8 Variabili di penalizzazione per la funzione obiettivo del caso in studio

L'introduzione delle variabili *di penalizzazione* $\{P_k$ MODEL BAD SEQUENCE FOR $R_j\}$ (con riferimento a ciascuna risorsa R_j del Resource Set) avviene per ogni processo P_k che nel proprio Albero degli Accessi presenti *potenzialmente almeno due figli*. *L'esistenza di due Processi Figli* costituisce infatti la condizione necessaria e sufficiente affinché il *processo padre P_k* abbia *richiesto l'accesso a due differenti risorse in istanti temporali diversi.*

Riprendendo il caso di studio in analisi, l'unico task che può disporre di tali variabili è P_1. Esse sono indicate come:

- $\{P_1$ MODEL BAD SEQUENCE FOR $R_h\}$, $\forall R_h \in$ Resource Set.

Il controllo del valore assunto da tali variabili sono avviene utilizzando il seguente and logico:

$$P_1 \, MODEL \, BAD \, SEQUENCE \, FOR \, R_h = \bigwedge \left\{ \begin{array}{c} REQUEST \, FOR \, R_h \, UNDER \, SON_1 \, OF \, P_1; \\ \neg \, DB \, ON \, R_h \, UNDER \, SON_1 \, OF \, P_1; \\ REQUEST \, FOR \, R_h \, UNDER \, SON_2 \, OF \, P_1; \\ DB \, ON \, R_h \, UNDER \, SON_2 \, OF \, P_1; \end{array} \right\}$$

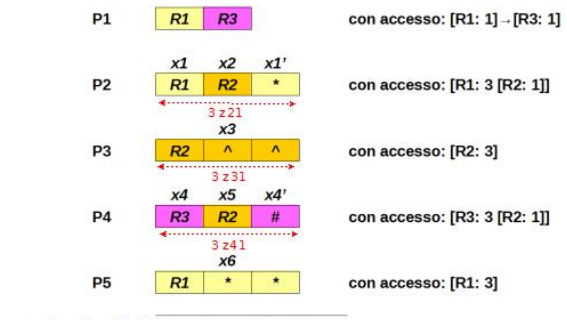
Nel paragrafo 4.4 di questo capitolo è stato inoltre detto che esse debbano comparire con coefficiente grandemente negativo in funzione obiettivo, così da *penalizzare la scelta di opportune soluzioni ottime*.

In conclusione, relativamente al Task Set in esame e nell'ipotesi che un *coefficiente sufficientemente negativo* sia "-1000" (esso è stato scelto in modo del tutto arbitrario), allora in F.O. risulterà presente la seguente *sommatoria di variabili di penalizzazione definite per P_1*:

- F.O. Max … - 1000 $\Sigma_{i=\{1,2,3\}}$ P_1 MODEL BAD SEQUENCE FOR R_i

4.9 Analisi dell'output fornito dal software di ottimizzazione GUROBI per il Modello ILP con vincoli temporali

```
# Objective value = 9
z21 1
z22 0
z23 0
z24 0
z31 1
z41 1
z42 0
z43 0
z44 0
z51 0

sequenza_non_ammessa_R1_AlberoP1 0
sequenza_non_ammessa_R2_AlberoP1 0
sequenza_non_ammessa_R3_AlberoP1 0

P1_richiede_R1 1
P1_ha_BD_su_R1 1
P1_NON_ha_BD_su_R1 0
P1_richiede_R3 1
P1_ha_BD_su_R3 1
P1_NON_ha_BD_su_R3 0

P2_utilizza_R1 1
P2_bloccante_per_R1 1
P2_rilascia_R1_per_P1 1
P2_richiede_R2 1
P2_ha_BD_su_R2 1
P2_NON_ha_BD_su_R2 0
P2_utilizza_R2 0
P2_bloccante_per_R2 0
P2_rilascia_R2_per_P4 0
```




```
P3_utilizza_R2 1
P3_bloccante_per_R2 1
P3_rilascia_R2_per_P2 1
P3_rilascia_R2_per_P4 0

P4_utilizza_R3 1
P4_bloccante_per_R3 1
P4_rilascia_R3_per_P1 1
P4_richiede_R2 1
P4_ha_BD_su_R2 0
P4_NON_ha_BD_su_R2 1
P4_utilizza_R2 0
P4_bloccante_per_R2 0
P4_rilascia_R2_per_P2 0

P5_utilizza_R1 0
P5_bloccante_per_R1 0
P5_rilascia_R1_per_P1 0
```

Figura 4.7 Output prodotto dal software di ottimizzazione GUROBI per il modello ILP del caso di studio in esame, con i vincoli temporali sinora discussi

Il paragrafo conclusivo dell'attuale capitolo è dedicato all'analisi del risultato sperimentale prodotto dal software di ottimizzazione GUROBI, nell'ipotesi che sia fornito il modello ILP dell'applicazione r-t sinora considerata, *con i vincoli temporali* discussi in precedenza.

Nell'immagine di sopra in realtà compaiono variabili binarie *mai incontrate* e qui adoperate solamente per fornire *nomi più intelligibili al lettore*, e che sono comunque facilmente associabili a quelle trattate in questo e nei capitoli addietro.

Quanto prodotto in uscita da GUROBI, conferma che:

- il valore del parametro B_{P1} è pari a 9 t.u., e determinato dalla presenza *in soluzione ottima* delle *variabili* z_{21}, z_{31}, z_{41} *con valore unitario*;
- la soluzione non vìola gli ordini di accesso alle risorse in quanto il blocco diretto per il possesso di R_2, causato da P_3, è subito da P_2;
- P_4 trova R_2 *già libera* al momento della propria richiesta d'accesso.

Un'ultima osservazione riguarda il fatto che quella illustrata è in realtà *una delle soluzioni ottime* al problema di *massimizzazione* del parametro B_{P1} per questo caso di studio.
È facile infatti notare come *l'entità* del tempo di blocco subito dal processo a maggior priorità sarebbe *identica* a quella appena calcolata, anche qualora P_1 subisse:

- un blocco diretto per R_1 e R_3 da parte di P_5 e P_4 rispettivamente, quindi uno *transitivo su* R_2 *via* P_4.

Questa soluzione *non infrange alcun ordine temporale* per quanto riguarda quelli con cui sono accedute le risorse dell'applicazione r-t, e non è fornita in uscita dal software di ottimizzazione semplicemente perché esso, secondo criteri arbitrari, sceglie di proporre quella descritta nella prima parte di questo paragrafo *a parità di valore della funzione obiettivo con quella ora in esame*.
Al contrario, quest'ultima sarebbe fornita *come unica soluzione ottima* se ad esempio l'estensione temporale della sezione critica x_6 di P_5 diventasse pari a 4 t.u. (ciò comporterebbe un incremento unitario dell'attuale coefficiente in f.o. della variabile z_{61}), definendo così un nuovo massimo globale per B_{P1}, in questo nuovo caso pari cioè a 10 t.u..
Il modello ILP del caso di studio è riportato *integralmente* con il nome *"*Modello ILP Applicazione r-t Fig.4.1 – Cap. 4*"* nel Capitolo V di questo elaborato.

CAPITOLO V

Prove sperimentali condotte per i casi di studio analizzati

5.1 Precisazioni sui test svolti

In questo capitolo conclusivo sono riportati i modelli ILP delle applicazioni real-time esaminate nei paragrafi precedenti.

Le soluzioni ottime discusse nei capitoli II, III e IV sono state ottenute adoperando il software di ottimizzazione GUROBI, versione 7.0.1, di proprietà di Gurobi Optimization Inc. (http://www.gurobi.com/), e fornendo ad esso in ingresso (in forma di file con estensione ".lp") i modelli di sotto esposti.

Lo scopo principale di questo capitolo finale è quello pertanto di fornire l'insieme di vincoli lineari *"sorgenti"* delle applicazioni analizzate, così da permettere al Lettore un'immediata verifica degli stessi attraverso un proprio *tool di ottimizzazione* (o quello di sopra proposto), ed avere un riscontro diretto della correttezza delle rappresentazioni ideate e delle corrispondenti soluzioni ottime.

Infine, per alcune applicazioni sono presenti *più funzioni obiettivo*, poiché la modifica dei coefficienti delle variabili z_k in f.o. e/o l'introduzione di alcune specifiche *variabili di modello*, consente di testare particolari proprietà e la robustezza delle modellazioni formulate.

5.2 Modello ILP "Applicazione r-t con esempio $P_k =_{def} P_3$ – Cap. 2"

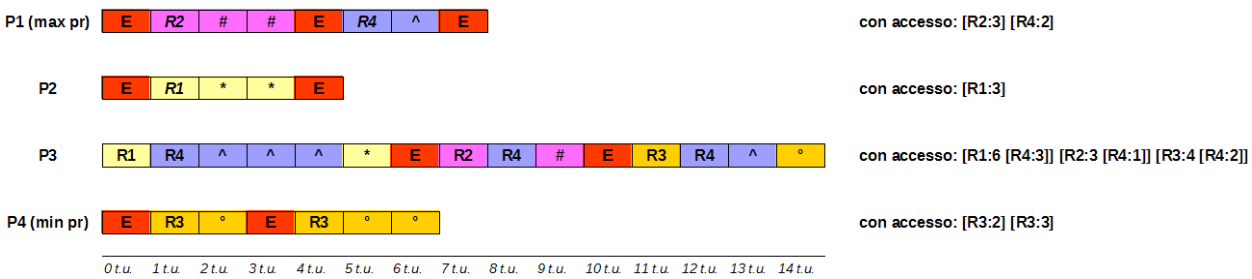


Figura 5.1 Applicazione r-t per il test delle proprietà del task $P_k =_{def} P_3$

Modello ILP:

```
\test1: z34a ottimo
\Max 3 z21 + 6 z31a + 5 z32a + z33a + 4 z34a + 3 z31b + 2 z32b + z33b + z34b + 4 z31c
+ 3 z32c + z33c + 2 z34c + 2 z41 + 3 z42 Subject To

\processo P1, con accesso alle risorse: [R2:3].[R4:2]
v1.P1: R2_REQUEST_LAUNCHED_BY_P1 + R4_REQUEST_LAUNCHED_BY_P1 = 2
```

```
\gestione R1_REQUEST_LAUNCHED_BY_P1
v2.P1: DB_ON_R2_FOR_P1 + NO_R2_DB_FOUND_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 = 0
v3.P1: R2_RELEASED_BY_P3_FOR_P1 - DB_ON_R2_FOR_P1 <= 0
v4.P1: R2_RELEASED_BY_P3_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 <= 0
v5.P1: R2_RELEASED_BY_P3_FOR_P1 - R2_REQUEST_MATCHED_BY_P3 <= 0
v6.P1: R2_RELEASED_BY_P3_FOR_P1 - DB_ON_R2_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 -
R2_REQUEST_MATCHED_BY_P3 >= - 2

v7.P1: DB_ON_R2_FOR_P1 - R2_RELEASED_BY_P3_FOR_P1 = 0

\gestione R4_REQUEST_LAUNCHED_BY_P1
v8.P1: DB_ON_R4_FOR_P1 + NO_R4_DB_FOUND_FOR_P1 - R4_REQUEST_LAUNCHED_BY_P1 = 0
v9.P1: R4_RELEASED_BY_P3_FOR_P1 - DB_ON_R4_FOR_P1 <= 0
v10.P1: R4_RELEASED_BY_P3_FOR_P1 - R4_REQUEST_LAUNCHED_BY_P1 <= 0
v11.P1: R4_RELEASED_BY_P3_FOR_P1 - R4_REQUEST_MATCHED_BY_P3 <= 0
v12.P1: R4_RELEASED_BY_P3_FOR_P1 - DB_ON_R4_FOR_P1 - R4_REQUEST_LAUNCHED_BY_P1 -
R4_REQUEST_MATCHED_BY_P3 >= - 2

v13.P1: DB_ON_R4_FOR_P1 - R4_RELEASED_BY_P3_FOR_P1 = 0
\FINE P1


\processo P2, con accesso alla risorsa: [R1:3].
v1.P2: z21 <= 1
v2.P2: z21 - R1_REQUEST_MATCHED_BY_P2 = 0
v3.P2: z21 - R1_MANAGED_BY_P2 = 0
\FINE P2


\P3
\1° processo fittizio introdotto per P3:
v1.P3.FCSS: z31a + z32a + z33a + z34a - P3_WILL_USE_R1_BY_z33a <= 1
v2.P3: z31a + z32a + z33a <= 1
v3.P3: z34a - P3_WILL_USE_R1_BY_z33a = 0
v4.P3: z31a + z34a <= 1
v5.P3: z32a + z34a <= 1
v6.P3: z31a + P3_WILL_USE_R1_BY_z33a <= 1
v7.P3: z32a + P3_WILL_USE_R1_BY_z33a <= 1
v8.P3: R1_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z31a - z32a - z33a
= 0
v9.P3: R4_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z34a = 0
v10.P3: R4_REQUEST_LAUNCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z31a = 0

v11.P3: R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z31a >= 0
v12.P3: R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z32a >= 0
v13.P3: R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z33a >= 0
v14.P3: R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z34a >= 0
v15.P3: R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z31a - z32a - z33a - z34a
<= 0
v16.P3: R4_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE - z32a - z34a = 0

\2° processo fittizio introdotto per P3:
v1.P3.SCSS: z31b + z32b + z33b + z34b - P3_WILL_USE_R2_BY_z33b <= 1
v2.P3.SCSS: z31b + z32b + z33b <= 1
v3.P3.SCSS: z34b - P3_WILL_USE_R2_BY_z33b = 0
v4.P3.SCSS: z31b + z34b <= 1
v5.P3.SCSS: z32b + z34b <= 1
v6.P3.SCSS: z31b + P3_WILL_USE_R2_BY_z33b <= 1
v7.P3.SCSS: z32b + P3_WILL_USE_R2_BY_z33b <= 1
v8.P3.SCSS: R2_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z31b - z32b -
z33b = 0
v9.P3.SCSS: R4_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z34b = 0
v10.P3.SCSS: R4_REQUEST_LAUNCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z31b = 0
v11.P3.SCSS: R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z31b >= 0
v12.P3.SCSS: R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z32b >= 0
v13.P3.SCSS: R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z33b >= 0
v14.P3.SCSS: R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z34b >= 0
```

```
v15.P3.SCSS: R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z31b - z32b - z33b -
z34b <= 0

v16.P3.SCSS: R4_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE - z32b - z34b = 0

\3° processo fittizio introdotto per P3:
v1.P3.TCSS: z31c + z32c + z33c + z34c - P3_WILL_USE_R3_BY_z33c <= 1
v2.P3.TCSS: z31c + z32c + z33c <= 1
v3.P3.TCSS: z34c - P3_WILL_USE_R3_BY_z33c = 0
v4.P3.TCSS: z31c + z34c <= 1
v5.P3.TCSS: z32c + z34c <= 1
v6.P3.TCSS: z31c + P3_WILL_USE_R3_BY_z33c <= 1
v7.P3.TCSS: z32c + P3_WILL_USE_R3_BY_z33c <= 1
v8.P3.TCSS: R3_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z31c - z32c -
z33c = 0
v9.P3.TCSS: R4_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z34c = 0
v10.P3.TCSS: R4_REQUEST_LAUNCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z31c = 0

v11.P3.TCSS: R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z31c >= 0
v12.P3.TCSS: R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z32c >= 0
v13.P3.TCSS: R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z33c >= 0
v14.P3.TCSS: R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z34c >= 0
v15.P3.TCSS: R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z31c - z32c - z33c -
z34c <= 0

v16.P3.TCSS: R4_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE - z32c - z34c = 0

\vincoli di definizione delle variabili globali di P3:
v1.P3.GLOBAL_VARs: R1_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE <= 1
v2.P3.GLOBAL_VARs: R1_REQUEST_MATCHED_BY_P3 -
R1_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE = 0

v3.P3.GLOBAL_VARs: R2_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE <= 1
v4.P3.GLOBAL_VARs: R2_REQUEST_MATCHED_BY_P3 -
R2_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE = 0

v5.P3.GLOBAL_VARs: R3_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE <= 1
v6.P3.GLOBAL_VARs: R3_REQUEST_MATCHED_BY_P3 -
R3_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE = 0

v7.P3.GLOBAL_VARs: R4_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE +
R4_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE +
R4_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE <= 1
v8.P3.GLOBAL_VARs: R4_REQUEST_MATCHED_BY_P3 -
R4_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE -
R4_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE -
R4_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE = 0

v9.P3.GLOBAL_VARs: R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE <= 1
v10.P3.GLOBAL_VARs: R1_MANAGED_BY_P3 -
R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE = 0

v11.P3.GLOBAL_VARs: R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE <= 1
v12.P3.GLOBAL_VARs: R2_MANAGED_BY_P3 -
R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE = 0

v13.P3.GLOBAL_VARs: R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE <= 1
v14.P3.GLOBAL_VARs: R3_MANAGED_BY_P3 -
R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE = 0

v15.P3.GLOBAL_VARs: R4_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE +
R4_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE +
R4_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE <= 1
v16.P3.GLOBAL_VARs: R4_MANAGED_BY_P3 -
R4_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE -
R4_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE -
R4_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE = 0
```

```
v17.P3.GLOBAL_VARs: R4_REQUEST_LAUNCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE +
R4_REQUEST_LAUNCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE +
R4_REQUEST_LAUNCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE <= 1
v18.P3.GLOBAL_VARs: R4_REQUEST_LAUNCHED_BY_P3 -
R4_REQUEST_LAUNCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE -
R4_REQUEST_LAUNCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE -
R4_REQUEST_LAUNCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE = 0

v1.P3: OR_zks_SET_1st_CRITICAL_SECTION_SEQUENCE - z31a >= 0
v2.P3: OR_zks_SET_1st_CRITICAL_SECTION_SEQUENCE - z32a >= 0
v3.P3: OR_zks_SET_1st_CRITICAL_SECTION_SEQUENCE - z33a >= 0
v4.P3: OR_zks_SET_1st_CRITICAL_SECTION_SEQUENCE - z34a >= 0
v5.P3: OR_zks_SET_1st_CRITICAL_SECTION_SEQUENCE - z31a - z32a - z33a - z34a <= 0

v6.P3: OR_zks_SET_2nd_CRITICAL_SECTION_SEQUENCE - z31b >= 0
v7.P3: OR_zks_SET_2nd_CRITICAL_SECTION_SEQUENCE - z32b >= 0
v8.P3: OR_zks_SET_2nd_CRITICAL_SECTION_SEQUENCE - z33b >= 0
v9.P3: OR_zks_SET_2nd_CRITICAL_SECTION_SEQUENCE - z34b >= 0
v10.P3: OR_zks_SET_2nd_CRITICAL_SECTION_SEQUENCE - z31b - z32b - z33b - z34b <= 0

v11.P3: OR_zks_SET_3rd_CRITICAL_SECTION_SEQUENCE - z31c >= 0
v12.P3: OR_zks_SET_3rd_CRITICAL_SECTION_SEQUENCE - z32c >= 0
v13.P3: OR_zks_SET_3rd_CRITICAL_SECTION_SEQUENCE - z33c >= 0
v14.P3: OR_zks_SET_3rd_CRITICAL_SECTION_SEQUENCE - z34c >= 0
v15.P3: OR_zks_SET_3rd_CRITICAL_SECTION_SEQUENCE - z31c - z32c - z33c - z34c <= 0

v16.P3: OR_zks_SET_1st_CRITICAL_SECTION_SEQUENCE +
OR_zks_SET_2nd_CRITICAL_SECTION_SEQUENCE + OR_zks_SET_3rd_CRITICAL_SECTION_SEQUENCE <=
1

v17.P3: R2_REQUEST_MATCHED_BY_P3 - R2_RELEASED_BY_P3_FOR_P1 <= 0
v18.P3: R4_REQUEST_MATCHED_BY_P3 - R4_RELEASED_BY_P3_FOR_P1 <= 0
\FINE P3

\processo P4, con accesso alla risorsa: [R3:2] -> [R3:3]
v1.P4: z41 + z42 <= 1
v2.P4: z41 + z42 - R3_REQUEST_MATCHED_BY_P4 = 0
v3.P4: z41 + z42 - R3_MANAGED_BY_P4 = 0
\FINE P4

\vincoli per la gestione della risorsa R1
v1: R1_REQUEST_MATCHED_BY_P2 + R1_REQUEST_MATCHED_BY_P3 <= 0

\vincoli per la gestione della risorsa R2
v1: R2_REQUEST_MATCHED_BY_P3 <= 1
v2: OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY - R2_REQUEST_MATCHED_BY_P3 = 0
v3: DB_ON_R2_FOR_P1 - OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY <= 0

v4: OR_RESULT_FOR_DB_ON_R2_PROPERTY - DB_ON_R2_FOR_P1 = 0
v5: R2_RELEASED_BY_P3_FOR_P1 - OR_RESULT_FOR_DB_ON_R2_PROPERTY <= 0

\vincoli per la gestione della risorsa R3
v6: R3_REQUEST_MATCHED_BY_P3 + R3_REQUEST_MATCHED_BY_P4 <= 0

\vincoli per la gestione della risorsa R4
v7: R4_REQUEST_MATCHED_BY_P3 <= 1
v8: OR_RESULT_FOR_R4_REQUEST_MATCHED_PROPERTY - R4_REQUEST_MATCHED_BY_P3 = 0
v9: DB_ON_R4_FOR_P1 - OR_RESULT_FOR_R4_REQUEST_MATCHED_PROPERTY <= 0

v10: OR_RESULT_FOR_DB_ON_R4_PROPERTY - DB_ON_R4_FOR_P1 = 0
v11: R4_RELEASED_BY_P3_FOR_P1 - OR_RESULT_FOR_DB_ON_R4_PROPERTY <= 0


Bounds

Binary
```

```
\P1 VARs
 R2_REQUEST_LAUNCHED_BY_P1
 R4_REQUEST_LAUNCHED_BY_P1

 DB_ON_R2_FOR_P1
 NO_R2_DB_FOUND_FOR_P1

 DB_ON_R4_FOR_P1
 NO_R4_DB_FOUND_FOR_P1
 \FINE P1 VARs

 \P2 VARs
 z21
 R1_REQUEST_MATCHED_BY_P2
 R1_MANAGED_BY_P2
 \FINE P2 VARs

 \P3 VARs
 R1_MANAGED_BY_P3
 R2_MANAGED_BY_P3
 R3_MANAGED_BY_P3
 R4_MANAGED_BY_P3

 R1_REQUEST_MATCHED_BY_P3
 R2_REQUEST_MATCHED_BY_P3
 R3_REQUEST_MATCHED_BY_P3
 R4_REQUEST_MATCHED_BY_P3

 R4_REQUEST_LAUNCHED_BY_P3
 OR_zks_SET_1st_CRITICAL_SECTION_SEQUENCE
 OR_zks_SET_2nd_CRITICAL_SECTION_SEQUENCE
 OR_zks_SET_3rd_CRITICAL_SECTION_SEQUENCE

 \P3.se1 VARS
      z31a
      z32a
      z33a
      z34a

      P3_WILL_USE_R1_BY_z33a
      R1_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE
      R4_REQUEST_MATCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE

      R4_REQUEST_LAUNCHED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE

      R1_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE
      R4_MANAGED_BY_P3_ON_1st_CRITICAL_SECTIONS_SEQUENCE

  \P3.se2 VARS
      z31b
      z32b
      z33b
      z34b

      P3_WILL_USE_R2_BY_z33b
      R2_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE
      R4_REQUEST_MATCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE

      R4_REQUEST_LAUNCHED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE

      R2_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE
      R4_MANAGED_BY_P3_ON_2nd_CRITICAL_SECTIONS_SEQUENCE

  \P3.se3 VARS
      z31c
      z32c
      z33c
      z34c
```

```
        P3_WILL_USE_R3_BY_z33c
        R3_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE
        R4_REQUEST_MATCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE

        R4_REQUEST_LAUNCHED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE

        R3_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE
        R4_MANAGED_BY_P3_ON_3rd_CRITICAL_SECTIONS_SEQUENCE


 R2_RELEASED_BY_P3_FOR_P1
 R4_RELEASED_BY_P3_FOR_P1
 \FINE P3 VARs

 \P4 VARs
 z41
 z42
 R3_REQUEST_MATCHED_BY_P4
 R3_MANAGED_BY_P4
 \FINE P4 VARs

 \R2, R4 VARs
 OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R2_PROPERTY

 OR_RESULT_FOR_R4_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R4_PROPERTY

End
```

5.3 Modello ILP "Applicazione r-t Fig. 3.5 – Cap. 3"

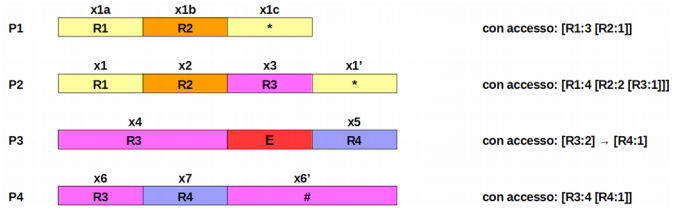


Figura 5.2 Applicazione r-t figura 3.5 del Capitolo III

Modello ILP:

```
Max 4 z21 + 3 z22 + 2 z23 + z24 + 2 z25 + z26a + z26b + 2 z31 + z32 + 4 z41 + 3 z42 +
2 z43 + z44 Subject To


\P1 (max pr), con accesso alle risorse: [R1:3 [R2:1]]
v1.P1: R1_REQUEST_LAUNCHED_BY_P1 + R2_REQUEST_LAUNCHED_BY_P1 = 2

\P1: gestione R1_REQUEST_LAUNCHED_BY_P1
v2.P1: DB_ON_R1_FOR_P1 + NO_R1_DB_FOUND_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 = 0
v3.P1: R1_RELEASED_BY_P2_FOR_P1 - DB_ON_R1_FOR_P1 <= 0
v4.P1: R1_RELEASED_BY_P2_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 <= 0
v5.P1: R1_RELEASED_BY_P2_FOR_P1 - R1_REQUEST_MATCHED_BY_P2 <= 0
```

```
v6.P1: R1_RELEASED_BY_P2_FOR_P1 - DB_ON_R1_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 -
R1_REQUEST_MATCHED_BY_P2 >= - 2
v7.P1: DB_ON_R1_FOR_P1 - R1_RELEASED_BY_P2_FOR_P1 = 0

\P1: gestione R2_REQUEST_LAUNCHED_BY_P1
v8.P1: DB_ON_R2_FOR_P1 + NO_R2_DB_FOUND_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 = 0
v9.P1: R2_RELEASED_BY_P2_FOR_P1 - DB_ON_R2_FOR_P1 <= 0
v10.P1: R2_RELEASED_BY_P2_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 <= 0
v11.P1: R2_RELEASED_BY_P2_FOR_P1 - R2_REQUEST_MATCHED_BY_P2 <= 0
v12.P1: R2_RELEASED_BY_P2_FOR_P1 - DB_ON_R2_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 -
R2_REQUEST_MATCHED_BY_P2 >= - 2

v13.P1: DB_ON_R2_FOR_P1 - R2_RELEASED_BY_P2_FOR_P1 = 0
\fine P1

\P2, con accesso alle risorse: [R1:4 [R2:2 [R3:1]]]
v1.P2: z21 + z22 + z23 + z24 + z25 + z26a + z26b - P2_WILL_USE_R1_BY_z24 <= 1

\mutua esclusione tra {z25, z26a, z26b} e le combinazioni zj di sezioni critiche che
rispondono alle richieste di R1:
v2.P2: z25 + z21 <= 1
v3.P2: z25 + z22 <= 1
v4.P2: z25 + z23 <= 1

v5.P2: z26a + z21 <= 1
v6.P2: z26a + z22 <= 1
v7.P2: z26a + z23 <= 1

v8.P2: z26b + z21 <= 1
v9.P2: z26b + z22 <= 1
v10.P2: z26b + z23 <= 1

\per Pi_WILL_USE_R1_BY_z4:
v11.P2: z21 + P2_WILL_USE_R1_BY_z24 <= 1
v12.P2: z22 + P2_WILL_USE_R1_BY_z24 <= 1
v13.P2: z23 + P2_WILL_USE_R1_BY_z24 <= 1

\mutua esclusione tra {z25, z26} ed equazione di definizione per Pi_WILL_USE_R1_BY_z4:
v14.P2: z25 + z26a + z26b <= 1
v15.P2: z25 + z26a + z26b - P2_WILL_USE_R1_BY_z24 = 0

\mutua esclusione tra {z21, z22, z23 e z24}:
v16.P2: z21 + z22 + z23 + z24 <= 1

\per le variabili di tipo {Rx_REQUEST_MATCHED_BY_P2}:
v17.P2: R1_REQUEST_MATCHED_BY_P2 - z21 - z22 - z23 - z24 = 0
```

```
v18.P2: R2_REQUEST_MATCHED_BY_P2 - z25 - z26a = 0
v19.P2: R3_REQUEST_MATCHED_BY_P2 - z26b = 0

\per la variabile R1_MANAGED_BY_P2:
v20.P2: R1_MANAGED_BY_P2 - z21 >= 0
v21.P2: R1_MANAGED_BY_P2 - z22 >= 0
v22.P2: R1_MANAGED_BY_P2 - z23 >= 0
v23.P2: R1_MANAGED_BY_P2 - z24 >= 0
v24.P2: R1_MANAGED_BY_P2 - z25 >= 0
v25.P2: R1_MANAGED_BY_P2 - z26a >= 0
v26.P2: R1_MANAGED_BY_P2 - z26b >= 0
v27.P2: R1_MANAGED_BY_P2 - z21 - z22 - z23 - z24 - z25 - z26a - z26b <= 0

\per la variabile R2_MANAGED_BY_P2:
v28.P2: R2_MANAGED_BY_P2 - z22 >= 0
v29.P2: R2_MANAGED_BY_P2 - z23 >= 0
v30.P2: R2_MANAGED_BY_P2 - z25 >= 0
v31.P2: R2_MANAGED_BY_P2 - z26a >= 0
v32.P2: R2_MANAGED_BY_P2 - z26b >= 0
v33.P2: R2_MANAGED_BY_P2 - z22 - z23 - z25 - z26a - z26b <= 0

\per la variabile R3_MANAGED_BY_P2:
v34.P2: R3_MANAGED_BY_P2 - z23 >= 0
v35.P2: R3_MANAGED_BY_P2 - z26a >= 0
v36.P2: R3_MANAGED_BY_P2 - z26b >= 0
v37.P2: R3_MANAGED_BY_P2 - z23 - z26a - z26b <= 0

\per la variabile R2_REQUEST_LAUNCHED_BY_P2:
v38.P2: R2_REQUEST_LAUNCHED_BY_P2 - z21 = 0

\per la variabile R3_REQUEST_LAUNCHED_BY_P2:
v39.P2: R3_REQUEST_LAUNCHED_BY_P2 - z21 - z22 - z25 = 0

\ vincoli per la gestione delle variabili
{R1_REQUEST_MATCHED,R2_REQUEST_MATCHED,R3_REQUEST_MATCHED_BY_P2} di P2:
v40.P2: R1_REQUEST_MATCHED_BY_P2 - R1_RELEASED_BY_P2_FOR_P1 <= 0
v41.P2: R2_REQUEST_MATCHED_BY_P2 - R2_RELEASED_BY_P2_FOR_P1 <= 0

\ vincoli per la gestione del DB su R2 da parte di P2:
v42.P2: DB_ON_R2_FOR_P2 + NO_R2_DB_FOUND_FOR_P2 - R2_REQUEST_LAUNCHED_BY_P2 = 0

v43.P2: DB_ON_R2_FOR_P2 = 0
\questo vincolo è SIGNIFICATIVO. In particolare l'unico processo che prevede
l'utilizzo della risorsa R2 oltre a P2, è quello P1 con pr1>p2, quindi
\NON BLOCCANTE PER P2. Non esistono pertanto task Pj tali che: "pr2 < prj <pr1", ne
con "prj < pr2".
```

```
\Di conseguenza: non esiste nemmeno alcuna variabile binaria nel modello del tipo
"R2_RELEASED_BY_Pj_FOR_P2". Ergo: è impossibile che P2 subisca
\un blocco per il possesso della risorsa R2.
\ vincoli per la gestione del DB su R3 da parte di P2:
v44.P2: DB_ON_R3_FOR_P2 + NO_R3_DB_FOUND_FOR_P2 - R3_REQUEST_LAUNCHED_BY_P2 = 0

v45.P2: R3_RELEASED_BY_P3_FOR_P2 - DB_ON_R3_FOR_P2 <= 0
v46.P2: R3_RELEASED_BY_P3_FOR_P2 - R3_REQUEST_LAUNCHED_BY_P2 <= 0
v47.P2: R3_RELEASED_BY_P3_FOR_P2 - R3_REQUEST_MATCHED_BY_P3 <= 0
v48.P2: R3_RELEASED_BY_P3_FOR_P2 - DB_ON_R3_FOR_P2 - R3_REQUEST_LAUNCHED_BY_P2 -
R3_REQUEST_MATCHED_BY_P3 >= - 2

v49.P2: R3_RELEASED_BY_P4_FOR_P2 - DB_ON_R3_FOR_P2 <= 0
v50.P2: R3_RELEASED_BY_P4_FOR_P2 - R3_REQUEST_LAUNCHED_BY_P2 <= 0
v51.P2: R3_RELEASED_BY_P4_FOR_P2 - R3_REQUEST_MATCHED_BY_P4 <= 0
v52.P2: R3_RELEASED_BY_P4_FOR_P2 - DB_ON_R3_FOR_P2 - R3_REQUEST_LAUNCHED_BY_P2 -
R3_REQUEST_MATCHED_BY_P4 >= - 2

v53.P2: DB_ON_R3_FOR_P2 - R3_RELEASED_BY_P3_FOR_P2 - R3_RELEASED_BY_P4_FOR_P2 = 0
\fine P2

\processo P3, con accesso alle risorse: [R2:2].[R4:1]
v1.P3: z31 + z32 <= 1
v2.P3: z31 - R3_REQUEST_MATCHED_BY_P3 = 0
v3.P3: z31 - R3_MANAGED_BY_P3 = 0

\vincoli per la gestione del DB su R2 da parte di P3:
v4.P3: R3_REQUEST_MATCHED_BY_P3 - R3_RELEASED_BY_P3_FOR_P2 <= 0 \ -
R3_RELEASED_BY_P3_FOR_P4 non ha senso

v5.P3: z32 - R4_REQUEST_MATCHED_BY_P3 = 0
v6.P3: z32 - R4_MANAGED_BY_P3 = 0

\vincoli per la gestione del DB su R4 da parte di P3:
v7.P3: R4_REQUEST_MATCHED_BY_P3 - R4_RELEASED_BY_P3_FOR_P4 <= 0
\FINE P3

\processo P4, con accesso alle risorse: [R3:3 [R4:1]]
v1.P4: z41 + z42 + z43 + z44 - P4_WILL_USE_R3_BY_z43 <= 1
v2.P4: z41 + z42 + z43 <= 1
v3.P4: z44 - P4_WILL_USE_R3_BY_z43 = 0
v4.P4: z41 + z44 <= 1
v5.P4: z42 + z44 <= 1
v6.P4: z41 + P4_WILL_USE_R3_BY_z43 <= 1
v7.P4: z42 + P4_WILL_USE_R3_BY_z43 <= 1
v8.P4: R3_REQUEST_MATCHED_BY_P4 - z41 - z42 - z43 = 0
```

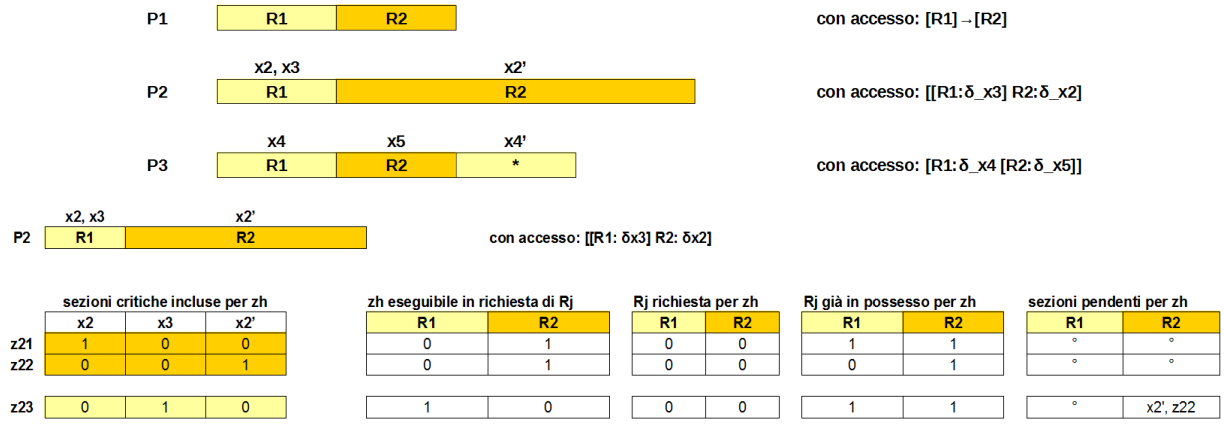
```
v9.P4: R4_REQUEST_MATCHED_BY_P4 - z44 = 0
v10.P4: R4_REQUEST_LAUNCHED_BY_P4 - z41 = 0
v11.P4: R3_MANAGED_BY_P4 - z41 >= 0
v12.P4: R3_MANAGED_BY_P4 - z42 >= 0
v13.P4: R3_MANAGED_BY_P4 - z43 >= 0
v14.P4: R3_MANAGED_BY_P4 - z44 >= 0
v15.P4: R3_MANAGED_BY_P4 - z41 - z42 - z43 - z44 <= 0
v16.P4: R4_MANAGED_BY_P4 - z42 - z44 = 0

\vincoli per la gestione delle variabili {R3_REQUEST_MATCHED_BY_P4,
R4_REQUEST_MATCHED_BY_P4} da parte di P4:
v17.P4: R3_REQUEST_MATCHED_BY_P4 - R3_RELEASED_BY_P4_FOR_P2 <= 0

\vincoli per la gestione del DB su R4 da parte di P4:
v18.P4: DB_ON_R4_FOR_P4 + NO_R4_DB_FOUND_FOR_P4 - R4_REQUEST_LAUNCHED_BY_P4 = 0
v19.P4: R4_RELEASED_BY_P3_FOR_P4 - DB_ON_R4_FOR_P4 <= 0
v20.P4: R4_RELEASED_BY_P3_FOR_P4 - R4_REQUEST_LAUNCHED_BY_P4 <= 0
v21.P4: R4_RELEASED_BY_P3_FOR_P4 - R4_REQUEST_MATCHED_BY_P3 <= 0
v22.P4: R4_RELEASED_BY_P3_FOR_P4 - DB_ON_R4_FOR_P4 - R4_REQUEST_LAUNCHED_BY_P4 -
R4_REQUEST_MATCHED_BY_P3 >= - 2
v23.P4: DB_ON_R4_FOR_P4 - R4_RELEASED_BY_P3_FOR_P4 = 0

\vincoli per la gestione della risorsa R1
v1.R1: R1_REQUEST_MATCHED_BY_P2 <= 1
v2.R1: OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY - R1_REQUEST_MATCHED_BY_P2 = 0
v3.R1: DB_ON_R1_FOR_P1 - OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY <= 0
v4.R1: OR_RESULT_FOR_DB_ON_R1_PROPERTY - DB_ON_R1_FOR_P1 = 0
v5.R1: R1_RELEASED_BY_P2_FOR_P1 - OR_RESULT_FOR_DB_ON_R1_PROPERTY <= 0
\FINE

\vincoli per la gestione della risorsa R2
v1.R2: R2_REQUEST_MATCHED_BY_P2 <= 1
v2.R2: OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY - R2_REQUEST_MATCHED_BY_P2 = 0
v3.R2: DB_ON_R2_FOR_P1 - OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY <= 0

v4.R2: OR_RESULT_FOR_DB_ON_R2_PROPERTY - DB_ON_R2_FOR_P1 = 0
v5.R2: R2_RELEASED_BY_P2_FOR_P1 - OR_RESULT_FOR_DB_ON_R2_PROPERTY <= 0
\FINE

\vincoli per la gestione della risorsa R3
v1.R3: R3_REQUEST_MATCHED_BY_P2 + R3_REQUEST_MATCHED_BY_P3 + R3_REQUEST_MATCHED_BY_P4
<= 1
v2.R3: OR_RESULT_FOR_R3_REQUEST_MATCHED_PROPERTY - R3_REQUEST_MATCHED_BY_P2 -
R3_REQUEST_MATCHED_BY_P3 - R3_REQUEST_MATCHED_BY_P4 = 0
v3.R3: DB_ON_R3_FOR_P2 - OR_RESULT_FOR_R3_REQUEST_MATCHED_PROPERTY <= 0
```

```
v4.R3: OR_RESULT_FOR_DB_ON_R3_PROPERTY - DB_ON_R3_FOR_P2 = 0
v5.R3: R3_RELEASED_BY_P3_FOR_P2 + R3_RELEASED_BY_P4_FOR_P2 -
OR_RESULT_FOR_DB_ON_R3_PROPERTY <= 0
\FINE

\vincoli per la gestione della risorsa R4
v1.R4: R4_REQUEST_MATCHED_BY_P3 + R4_REQUEST_MATCHED_BY_P4 <= 1
v2.R4: OR_RESULT_FOR_R4_REQUEST_MATCHED_PROPERTY - R4_REQUEST_MATCHED_BY_P3 -
R4_REQUEST_MATCHED_BY_P4 = 0
v3.R4: DB_ON_R4_FOR_P4 - OR_RESULT_FOR_R4_REQUEST_MATCHED_PROPERTY <= 0

v4.R4: OR_RESULT_FOR_DB_ON_R4_PROPERTY - DB_ON_R4_FOR_P4 = 0
v5.R4: R4_RELEASED_BY_P3_FOR_P4 - OR_RESULT_FOR_DB_ON_R4_PROPERTY <= 0

v1.MAIN: R3_MANAGED_BY_P2 + R3_MANAGED_BY_P3 + R3_MANAGED_BY_P4 <= 1
v2.MAIN: R4_MANAGED_BY_P3 + R4_MANAGED_BY_P4 <= 1

\Non è necessario introdurre i vincoli di tipo "HprP" per il processo P2, dato che:
\      ° l'unica risorsa condivisa tra questo stesso e i processi con pr<pr2 è quella
R3, ma tuttavia non esiste nel task set alcun processo
\      con pr>pr2 che la richieda (nota: vedi caso di P3, che può escludere P4 dal lock
di R3, in favore di P2).

\PER P3
v1.HprP.P3: AND_HIGHERprPROCS_FOR_z32 - z32 <= 0
v2.HprP.P3: AND_HIGHERprPROCS_FOR_z32 + z31 <= 1
v3.HprP.P3: AND_HIGHERprPROCS_FOR_z32 + z31 - z32 >= 0

\vincoli di definzione di R3_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr3
v4.HprP.P3: R3_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr3 -
AND_HIGHERprPROCS_FOR_z32 = 0

v5.HprP.P3: R3_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr3 + R3_MANAGED_BY_P3 +
R3_MANAGED_BY_P4 <= 1

Bounds

Binary

 \P1 VARs
 R1_REQUEST_LAUNCHED_BY_P1
 DB_ON_R1_FOR_P1
 NO_R1_DB_FOUND_FOR_P1
 R2_REQUEST_LAUNCHED_BY_P1
 DB_ON_R2_FOR_P1
 NO_R2_DB_FOUND_FOR_P1
```

```
 \P2 VARs
z21
z22
z23
z24
z25
z26a
z26b
P2_WILL_USE_R1_BY_z24
R1_MANAGED_BY_P2
R2_MANAGED_BY_P2
R3_MANAGED_BY_P2
R1_REQUEST_MATCHED_BY_P2
R2_REQUEST_MATCHED_BY_P2
R3_REQUEST_MATCHED_BY_P2
R2_REQUEST_LAUNCHED_BY_P2
DB_ON_R2_FOR_P2
NO_R2_DB_FOUND_FOR_P2
R3_REQUEST_LAUNCHED_BY_P2
DB_ON_R3_FOR_P2
NO_R3_DB_FOUND_FOR_P2
R1_RELEASED_BY_P2_FOR_P1
R2_RELEASED_BY_P2_FOR_P1

\P3 VARs
z31
R3_REQUEST_MATCHED_BY_P3
R3_MANAGED_BY_P3
R3_RELEASED_BY_P3_FOR_P2
z32
R4_REQUEST_MATCHED_BY_P3
R4_MANAGED_BY_P3
R4_RELEASED_BY_P3_FOR_P4

\P4 VARs
z41
z42
z43
z44
P4_WILL_USE_R3_BY_z43
R3_MANAGED_BY_P4
R4_MANAGED_BY_P4
R3_REQUEST_MATCHED_BY_P4
R4_REQUEST_MATCHED_BY_P4
R4_REQUEST_LAUNCHED_BY_P4
DB_ON_R4_FOR_P4
```

```
 NO_R4_DB_FOUND_FOR_P4
 R3_RELEASED_BY_P4_FOR_P2

 \R1 VARs
 OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R1_PROPERTY

 \R2 VARs
 OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R2_PROPERTY

 \R3 VARs
 OR_RESULT_FOR_R3_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R3_PROPERTY

 \R4 VARs
 OR_RESULT_FOR_R4_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R4_PROPERTY

 \P3 HigherprProcs VARs
  AND_HIGHERprPROCS_FOR_z32
  R3_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr3
End
```

5.4 Modello ILP "Applicazione r-t Fig. 3.7 – Cap. 3"



	sezioni critiche incluse per zh		
	x2	x3	x2'
z21	1	0	0
z22	0	0	1
z23	0	1	0

zh eseguibile in richiesta di Rj	
R1	R2
0	1
0	1
1	0

Rj richiesta per zh	
R1	R2
0	0
0	0
0	0

Rj già in possesso per zh	
R1	R2
1	1
0	1
1	1

sezioni pendenti per zh	
R1	R2
°	°
°	°
°	x2', z22

nota:

per il test 1: $(\delta x_3, \delta x_2) = (3, 4)$, $(\delta x_4, \delta x_5) = (3,1)$;

per il test 2: $(\delta x_3, \delta x_2) = (3, 4)$, $(\delta x_4, \delta x_5) = (5,1)$;

Figura 5.3 Applicazione r-t figura 3.7 del Capitolo III e tabella delle proprietà di P_2

Modello ILP:

```
\test1:
\Maximize 4 z21 + 3 z22 + 1 z23 + 3 z31 + 2 z32 + 1 z33 + 1 z34 Subject To

\test2:
Maximize 4 z21 + 3 z22 + 1 z23 + 5 z31 + 2 z32 + 1 z33 + 1 z34 Subject To

\processo P1 (max pr), con accesso alle risorse: [R1:1] -> [R2:1]
vP1.0: R1_REQUEST_LAUNCHED_BY_P1 + R2_REQUEST_LAUNCHED_BY_P1 = 2

\ vincoli per la gestione del DB su R1 da parte di P1:
vP1.1: DB_ON_R1_FOR_P1 + NO_R1_DB_FOUND_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 = 0
vP1.2: R1_RELEASED_BY_P2_FOR_P1 - DB_ON_R1_FOR_P1 <= 0
vP1.3: R1_RELEASED_BY_P2_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 <= 0
vP1.4: R1_RELEASED_BY_P2_FOR_P1 - R1_REQUEST_MATCHED_BY_P2 <= 0
vP1.5: R1_RELEASED_BY_P2_FOR_P1 - DB_ON_R1_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 -
R1_REQUEST_MATCHED_BY_P2 >= - 2

vP1.6: R1_RELEASED_BY_P3_FOR_P1 - DB_ON_R1_FOR_P1 <= 0
vP1.7: R1_RELEASED_BY_P3_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 <= 0
vP1.8: R1_RELEASED_BY_P3_FOR_P1 - R1_REQUEST_MATCHED_BY_P3 <= 0
vP1.9: R1_RELEASED_BY_P3_FOR_P1 - DB_ON_R1_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 -
R1_REQUEST_MATCHED_BY_P3 >= - 2

vP1.10: DB_ON_R1_FOR_P1 - R1_RELEASED_BY_P2_FOR_P1 - R1_RELEASED_BY_P3_FOR_P1 = 0

\ vincoli per la gestione del DB su R2 da parte di P1:
vP1.11: DB_ON_R2_FOR_P1 + NO_R2_DB_FOUND_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 = 0
vP1.12: R2_RELEASED_BY_P2_FOR_P1 - DB_ON_R2_FOR_P1 <= 0
vP1.13: R2_RELEASED_BY_P2_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 <= 0
vP1.14: R2_RELEASED_BY_P2_FOR_P1 - R2_REQUEST_MATCHED_BY_P2 <= 0
vP1.15: R2_RELEASED_BY_P2_FOR_P1 - DB_ON_R2_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 -
R2_REQUEST_MATCHED_BY_P2 >= - 2

vP1.16: R2_RELEASED_BY_P3_FOR_P1 - DB_ON_R2_FOR_P1 <= 0
vP1.17: R2_RELEASED_BY_P3_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 <= 0
vP1.18: R2_RELEASED_BY_P3_FOR_P1 - R2_REQUEST_MATCHED_BY_P3 <= 0
vP1.19: R2_RELEASED_BY_P3_FOR_P1 - DB_ON_R2_FOR_P1 - R2_REQUEST_LAUNCHED_BY_P1 -
R2_REQUEST_MATCHED_BY_P3 >= - 2

vP1.20: DB_ON_R2_FOR_P1 - R2_RELEASED_BY_P2_FOR_P1 - R2_RELEASED_BY_P3_FOR_P1 = 0
\FINE
```

```
\processo P2, con accesso alle risorse: [[R1:delta_x3] R2:delta_x2]
vP2.1: z21 + z22 + z23 - P2_WILL_USE_R2_BY_z22 <= 1
vP2.2: z21 + z22 <= 1
vP2.3: z23 - P2_WILL_USE_R2_BY_z22 = 0
vP2.4: z21 + P2_WILL_USE_R2_BY_z22 <= 1
vP2.5: z21 + z23 <= 1
vP2.6: R1_REQUEST_MATCHED_BY_P2 - z23 = 0
vP2.7: R2_REQUEST_MATCHED_BY_P2 - z21 - z22 = 0
vP2.8: R1_MANAGED_BY_P2 - z21 - z23 = 0
vP2.9: R2_MANAGED_BY_P2 - z21 >= 0
vP2.10: R2_MANAGED_BY_P2 - z22 >= 0
vP2.11: R2_MANAGED_BY_P2 - z23 >= 0
vP2.12: R2_MANAGED_BY_P2 - z21 - z22 - z23 <= 0

vP2.13: AND_HIGHERprPROCS_FOR_z22 - z22 <= 0
vP2.14: AND_HIGHERprPROCS_FOR_z22 + z21 <= 1
vP2.15: AND_HIGHERprPROCS_FOR_z22 + z23 <= 1
vP2.16: AND_HIGHERprPROCS_FOR_z22 + z21 + z23 - z22 >= 0

vP2.17: R1_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr2 - AND_HIGHERprPROCS_FOR_z22
= 0

vP2.18: R1_REQUEST_MATCHED_BY_P2 - R1_RELEASED_BY_P2_FOR_P1 <= 0
vP2.19: R2_REQUEST_MATCHED_BY_P2 - R2_RELEASED_BY_P2_FOR_P1 - R2_RELEASED_BY_P2_FOR_P3
<= 0

\processo P3 (min pr), con accesso alle risorse: [R1:3 [R2:1]]
vP3.1: z31 + z32 + z33 + z34 - P3_WILL_USE_R1_BY_z33 <= 1
vP3.2: z31 + z32 + z33 <= 1
vP3.3: z34 - P3_WILL_USE_R1_BY_z33 = 0
vP3.4: z31 + z34 <= 1
vP3.5: z32 + z34 <= 1
vP3.6: z31 + P3_WILL_USE_R1_BY_z33 <= 1
vP3.7: z32 + P3_WILL_USE_R1_BY_z33 <= 1
vP3.8: R1_REQUEST_MATCHED_BY_P3 - z31 - z32 - z33 = 0
vP3.9: R2_REQUEST_MATCHED_BY_P3 - z34 = 0
vP3.10: R2_REQUEST_LAUNCHED_BY_P3 - z31 = 0
vP3.11: R1_MANAGED_BY_P3 - z31 >= 0
vP3.12: R1_MANAGED_BY_P3 - z32 >= 0
vP3.13: R1_MANAGED_BY_P3 - z33 >= 0
vP3.14: R1_MANAGED_BY_P3 - z34 >= 0
vP3.15: R1_MANAGED_BY_P3 - z31 - z32 - z33 - z34 <= 0
vP3.16: R2_MANAGED_BY_P3 - z32 - z34 = 0

\ vincoli per la gestione del DB su R2 da parte di P2:
vP3.20: DB_ON_R2_FOR_P3 + NO_R2_DB_FOUND_FOR_P3 - R2_REQUEST_LAUNCHED_BY_P3 = 0
```

```
vP3.21: R2_RELEASED_BY_P2_FOR_P3 - DB_ON_R2_FOR_P3 <= 0
vP3.22: R2_RELEASED_BY_P2_FOR_P3 - R2_REQUEST_LAUNCHED_BY_P3 <= 0
vP3.23: R2_RELEASED_BY_P2_FOR_P3 - R2_REQUEST_MATCHED_BY_P2 <= 0
vP3.24: R2_RELEASED_BY_P2_FOR_P3 - DB_ON_R2_FOR_P3 - R2_REQUEST_LAUNCHED_BY_P3 -
R2_REQUEST_MATCHED_BY_P2 >= - 2

vP3.25: DB_ON_R2_FOR_P3 - R2_RELEASED_BY_P2_FOR_P3 = 0

vP3.26: R1_REQUEST_MATCHED_BY_P3 - R1_RELEASED_BY_P3_FOR_P1 <= 0
vP3.27: R2_REQUEST_MATCHED_BY_P3 - R2_RELEASED_BY_P3_FOR_P1 <= 0

\vincoli per la gestione della risorsa R1
v1: R1_REQUEST_MATCHED_BY_P2 + R1_REQUEST_MATCHED_BY_P3 <= 1
v2: OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY - R1_REQUEST_MATCHED_BY_P2 -
R1_REQUEST_MATCHED_BY_P3 = 0
v3: DB_ON_R1_FOR_P1 - OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY <= 0

v4: OR_RESULT_FOR_DB_ON_R1_PROPERTY - DB_ON_R1_FOR_P1 = 0
v5: R1_RELEASED_BY_P2_FOR_P1 + R1_RELEASED_BY_P3_FOR_P1 -
OR_RESULT_FOR_DB_ON_R1_PROPERTY <= 0

\vincoli per la gestione della risorsa R2
v6: R2_REQUEST_MATCHED_BY_P2 + R2_REQUEST_MATCHED_BY_P3 <= 1
v7: OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY - R2_REQUEST_MATCHED_BY_P2 -
R2_REQUEST_MATCHED_BY_P3 = 0
v8: DB_ON_R2_FOR_P1 + DB_ON_R2_FOR_P3 - OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY <= 0

v9: OR_RESULT_FOR_DB_ON_R2_PROPERTY - DB_ON_R2_FOR_P1 - DB_ON_R2_FOR_P3 = 0
v10: R2_RELEASED_BY_P2_FOR_P1 + R2_RELEASED_BY_P2_FOR_P3 + R2_RELEASED_BY_P3_FOR_P1 -
OR_RESULT_FOR_DB_ON_R2_PROPERTY <= 0

v11: R1_MANAGED_BY_P2 + R1_MANAGED_BY_P3 <= 1
v12: R2_MANAGED_BY_P2 + R2_MANAGED_BY_P3 <= 1
v13: R1_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr2 + R1_MANAGED_BY_P2 +
R1_MANAGED_BY_P3 <= 1

Bounds

Binary

 \P1 VARs
 R1_REQUEST_LAUNCHED_BY_P1
 DB_ON_R1_FOR_P1
 NO_R1_DB_FOUND_FOR_P1
 R2_REQUEST_LAUNCHED_BY_P1
 DB_ON_R2_FOR_P1
```

```
 NO_R2_DB_FOUND_FOR_P1

 \P2 VARs
 z21
 z22
 z23
 R1_REQUEST_MATCHED_BY_P2
 R2_REQUEST_MATCHED_BY_P2
 R1_MANAGED_BY_P2
 R2_MANAGED_BY_P2
 P2_WILL_USE_R2_BY_z22
 R1_RELEASED_BY_P2_FOR_P1
 R2_RELEASED_BY_P2_FOR_P1
 R2_RELEASED_BY_P2_FOR_P3
 AND_HIGHERprPROCS_FOR_z22
 R1_LOCK_AVALAIBLE_FOR_Pj_WITH_prj_HIGHER_THAN_pr2

 \P3 VARs
 z31
 z32
 z33
 z34
 R1_MANAGED_BY_P3
 R1_REQUEST_MATCHED_BY_P3
 P3_WILL_USE_R1_BY_z33
 R2_REQUEST_LAUNCHED_BY_P3
 R2_MANAGED_BY_P3
 R2_REQUEST_MATCHED_BY_P3
 DB_ON_R2_FOR_P3
 NO_R2_DB_FOUND_FOR_P3
 R1_RELEASED_BY_P3_FOR_P1
 R2_RELEASED_BY_P3_FOR_P1

 OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R1_PROPERTY

 OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R2_PROPERTY
End
```

5.5 Modello ILP “Applicazione r-t Fig.4.1 – Cap. 4”

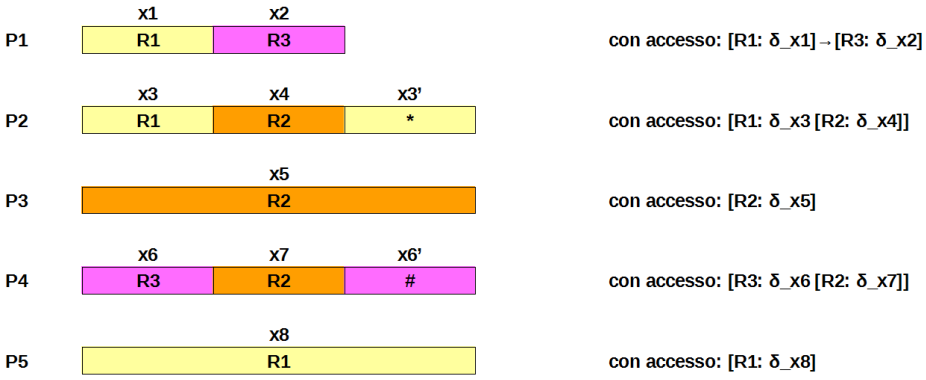


Figura 5.4 Applicazione r-t figura 4.1 del Capitolo IV

Modello ILP:

```
\test1: utile per testare le relazioni:
\      a) {'z21=1' => 'z51=0'};
\      b) {'R2_REQUEST_LAUNCHED_BY_P2=1' => 'R2_REQUEST_MATCHED_BY_P3=1' =>
'DB_ON_R2_FOR_P2=1'};
\      c) {'R2_REQUEST_LAUNCHED_BY_P4=1' => 'NO_R2_DB_FOUND_FOR_P4=1'};
\Max 10 z21 + z22 + z23 + z24 + z31 + 5 z41 + z42 + z43 + z44 + z51 + DB_ON_R2_FOR_P2
Subject To

\test2
\      a) {'z21=1' => 'z51=0'};
\      b) {'R2_REQUEST_LAUNCHED_BY_P4=1' => 'R2_REQUEST_MATCHED_BY_P3=1' =>
'DB_ON_R2_FOR_P4=1'};
\      c) {'R2_REQUEST_LAUNCHED_BY_P2=1' => 'NO_R2_DB_FOUND_FOR_P2=1'};
\Max 10 z21 + z22 + z23 + z24 + z31 + 5 z41 + z42 + z43 + z44 + z51 + DB_ON_R2_FOR_P4
Subject To

\test3: utile per testare le relazioni:
\      b) {'z51=1' => 'z2i=0'};
\      a) {'R2_REQUEST_LAUNCHED_BY_P4=1' => 'R2_REQUEST_MATCHED_BY_P3=1' =>
'DB_ON_R2_FOR_P4=1'}.
\Max z21 + z22 + z23 + z24 + z31 + 10 z41 + z42 + z43 + z44 + 20 z51 Subject To

\test4: utile per testare le relazioni:
\      a) {'z21=1' => 'z51=0'};
\      b) {'R2_REQUEST_LAUNCHED_BY_P2=1','R2_REQUEST_LAUNCHED_BY_P4=1' =>
'R2_IS_LOCKED=0' => 'NO_R2_DB_FOUND_FOR_P2=1,NO_R2_DB_FOUND_FOR_P4=1'}.
\Max 10 z21 + z22 + z23 + z24 - z31 + 10 z41 + z42 + z43 + z44 + z51 Subject To
```

```
\test5: utile per testare le relazioni:
Max 3 z21 + 2 z22 + z23 + z24 + 3 z31 + 3 z41 + 2 z42 + z43 + z44 + 3 z51 - 1000
P1_MODEL_BAD_SEQUENCE_FOR_R1 - 1000 P1_MODEL_BAD_SEQUENCE_FOR_R2 - 1000
P1_MODEL_BAD_SEQUENCE_FOR_R3 Subject To

\P1 (max pr), con accesso alle risorse: [R1].[R3]
v1: R1_REQUEST_LAUNCHED_BY_P1 + R3_REQUEST_LAUNCHED_BY_P1 = 2

\P1: gestione R1_REQUEST_LAUNCHED_BY_P1
v2: DB_ON_R1_FOR_P1 + NO_R1_DB_FOUND_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 = 0

v3: R1_RELEASED_BY_P2_FOR_P1 - DB_ON_R1_FOR_P1 <= 0
v4: R1_RELEASED_BY_P2_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 <= 0
v5: R1_RELEASED_BY_P2_FOR_P1 - R1_REQUEST_MATCHED_BY_P2 <= 0
v6: R1_RELEASED_BY_P2_FOR_P1 - DB_ON_R1_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 -
R1_REQUEST_MATCHED_BY_P2 >= - 2

v7: R1_RELEASED_BY_P5_FOR_P1 - DB_ON_R1_FOR_P1 <= 0
v8: R1_RELEASED_BY_P5_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 <= 0
v9: R1_RELEASED_BY_P5_FOR_P1 - R1_REQUEST_MATCHED_BY_P5 <= 0
v10: R1_RELEASED_BY_P5_FOR_P1 - DB_ON_R1_FOR_P1 - R1_REQUEST_LAUNCHED_BY_P1 -
R1_REQUEST_MATCHED_BY_P5 >= - 2

v11: DB_ON_R1_FOR_P1 - R1_RELEASED_BY_P2_FOR_P1 - R1_RELEASED_BY_P5_FOR_P1 = 0


\P1: gestione R1_REQUEST_LAUNCHED_BY_P1
v12: DB_ON_R3_FOR_P1 + NO_R3_DB_FOUND_FOR_P1 - R3_REQUEST_LAUNCHED_BY_P1 = 0

v13: R3_RELEASED_BY_P4_FOR_P1 - DB_ON_R3_FOR_P1 <= 0
v14: R3_RELEASED_BY_P4_FOR_P1 - R3_REQUEST_LAUNCHED_BY_P1 <= 0
v15: R3_RELEASED_BY_P4_FOR_P1 - R3_REQUEST_MATCHED_BY_P4 <= 0
v16: R3_RELEASED_BY_P4_FOR_P1 - DB_ON_R3_FOR_P1 - R3_REQUEST_LAUNCHED_BY_P1 -
R3_REQUEST_MATCHED_BY_P4 >= - 2

v17: DB_ON_R3_FOR_P1 - R3_RELEASED_BY_P4_FOR_P1 = 0

\ +++ P1 Vincoli per l'Albero degli Accessi (vedi TTCs: "Time Tree Constraints") +++
vP1.TTCs.1: REQUEST_FOR_R1_WHEN_P1_IS_FATHER - R1_REQUEST_LAUNCHED_BY_P1 >= 0
vP1.TTCs.2: REQUEST_FOR_R1_WHEN_P1_IS_FATHER - REQUEST_FOR_R1_UNDER_SON1_OF_P1 >= 0
vP1.TTCs.3: REQUEST_FOR_R1_WHEN_P1_IS_FATHER - REQUEST_FOR_R1_UNDER_SON2_OF_P1 >= 0
vP1.TTCs.4: REQUEST_FOR_R1_WHEN_P1_IS_FATHER - R1_REQUEST_LAUNCHED_BY_P1 -
REQUEST_FOR_R1_UNDER_SON1_OF_P1 - REQUEST_FOR_R1_UNDER_SON2_OF_P1 <= 0

vP1.TTCs.5: DB_ON_R1_WHEN_P1_IS_FATHER - DB_ON_R1_FOR_P1 >= 0
vP1.TTCs.6: DB_ON_R1_WHEN_P1_IS_FATHER - DB_ON_R1_UNDER_SON1_OF_P1 >= 0
```

```
vP1.TTCs.7: DB_ON_R1_WHEN_P1_IS_FATHER - DB_ON_R1_UNDER_SON2_OF_P1 >= 0
vP1.TTCs.8: DB_ON_R1_WHEN_P1_IS_FATHER - DB_ON_R1_FOR_P1 - DB_ON_R1_UNDER_SON1_OF_P1 -
DB_ON_R1_UNDER_SON2_OF_P1 <= 0

vP1.TTCs.9: REQUEST_FOR_R2_WHEN_P1_IS_FATHER - REQUEST_FOR_R2_UNDER_SON1_OF_P1 >= 0
vP1.TTCs.10: REQUEST_FOR_R2_WHEN_P1_IS_FATHER - REQUEST_FOR_R2_UNDER_SON2_OF_P1 >= 0
vP1.TTCs.11: REQUEST_FOR_R2_WHEN_P1_IS_FATHER - REQUEST_FOR_R2_UNDER_SON1_OF_P1 -
REQUEST_FOR_R2_UNDER_SON2_OF_P1 <= 0

vP1.TTCs.12: DB_ON_R2_WHEN_P1_IS_FATHER - DB_ON_R2_UNDER_SON1_OF_P1 >= 0
vP1.TTCs.13: DB_ON_R2_WHEN_P1_IS_FATHER - DB_ON_R2_UNDER_SON2_OF_P1 >= 0
vP1.TTCs.14: DB_ON_R2_WHEN_P1_IS_FATHER - DB_ON_R2_UNDER_SON1_OF_P1 -
DB_ON_R2_UNDER_SON2_OF_P1 <= 0

vP1.TTCs.15: REQUEST_FOR_R3_WHEN_P1_IS_FATHER - R3_REQUEST_LAUNCHED_BY_P1 >= 0
vP1.TTCs.16: REQUEST_FOR_R3_WHEN_P1_IS_FATHER - REQUEST_FOR_R3_UNDER_SON1_OF_P1 >= 0
vP1.TTCs.17: REQUEST_FOR_R3_WHEN_P1_IS_FATHER - REQUEST_FOR_R3_UNDER_SON2_OF_P1 >= 0
vP1.TTCs.18: REQUEST_FOR_R3_WHEN_P1_IS_FATHER - R3_REQUEST_LAUNCHED_BY_P1 -
REQUEST_FOR_R3_UNDER_SON1_OF_P1 - REQUEST_FOR_R3_UNDER_SON2_OF_P1 <= 0

vP1.TTCs.19: DB_ON_R3_WHEN_P1_IS_FATHER - DB_ON_R3_FOR_P1 >= 0
vP1.TTCs.20: DB_ON_R3_WHEN_P1_IS_FATHER - DB_ON_R3_UNDER_SON1_OF_P1 >= 0
vP1.TTCs.21: DB_ON_R3_WHEN_P1_IS_FATHER - DB_ON_R3_UNDER_SON2_OF_P1 >= 0
vP1.TTCs.22: DB_ON_R3_WHEN_P1_IS_FATHER - DB_ON_R3_FOR_P1 - DB_ON_R3_UNDER_SON1_OF_P1
- DB_ON_R3_UNDER_SON2_OF_P1 <= 0

vP1.TTCs.23: NO_SON1_FOR_P1 + DB_ON_R1_FOR_P1 = 1
vP1.TTCs.24: NO_SON2_FOR_P1 + DB_ON_R3_FOR_P1 = 1

\vincoli per NO_SON1_FOR_P1:
vP1.TTCs.25: REQUEST_FOR_R1_UNDER_SON1_OF_P2 + NO_SON1_FOR_P1 <= 1
vP1.TTCs.26: REQUEST_FOR_R2_UNDER_SON1_OF_P2 + NO_SON1_FOR_P1 <= 1
vP1.TTCs.27: REQUEST_FOR_R3_UNDER_SON1_OF_P2 + NO_SON1_FOR_P1 <= 1

vP1.TTCs.28: DB_ON_R1_UNDER_SON1_OF_P2 + NO_SON1_FOR_P1 <= 1
vP1.TTCs.29: DB_ON_R2_UNDER_SON1_OF_P2 + NO_SON1_FOR_P1 <= 1
vP1.TTCs.30: DB_ON_R3_UNDER_SON1_OF_P2 + NO_SON1_FOR_P1 <= 1

\vincoli per NO_SON2_FOR_P1:
vP1.TTCs.31: REQUEST_FOR_R1_UNDER_SON2_OF_P1 + NO_SON2_FOR_P1 <= 1
vP1.TTCs.32: REQUEST_FOR_R2_UNDER_SON2_OF_P1 + NO_SON2_FOR_P1 <= 1
vP1.TTCs.33: REQUEST_FOR_R3_UNDER_SON2_OF_P1 + NO_SON2_FOR_P1 <= 1

vP1.TTCs.34: DB_ON_R1_UNDER_SON2_OF_P1 + NO_SON2_FOR_P1 <= 1
vP1.TTCs.35: DB_ON_R2_UNDER_SON2_OF_P1 + NO_SON2_FOR_P1 <= 1
vP1.TTCs.36: DB_ON_R3_UNDER_SON2_OF_P1 + NO_SON2_FOR_P1 <= 1
```

```
\REQUEST_FOR_R1_UNDER_SON1_OF_P1
vP1.TTCs.37: AND_son1_P2_rqR1_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 <= 0
vP1.TTCs.38: AND_son1_P2_rqR1_FOR_P1_TTCs - REQUEST_FOR_R1_WHEN_P2_IS_FATHER <= 0
vP1.TTCs.39: AND_son1_P2_rqR1_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 -
REQUEST_FOR_R1_WHEN_P2_IS_FATHER >= - 1

vP1.TTCs.40: AND_son1_P5_rqR1_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 <= 0
vP1.TTCs.41: AND_son1_P5_rqR1_FOR_P1_TTCs - REQUEST_FOR_R1_WHEN_P5_IS_FATHER <= 0
vP1.TTCs.42: AND_son1_P5_rqR1_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 -
REQUEST_FOR_R1_WHEN_P5_IS_FATHER >= - 1

vP1.TTCs.43a: AND_son1_P2_rqR1_FOR_P1_TTCs + AND_son1_P5_rqR1_FOR_P1_TTCs <= 1
vP1.TTCs.43b: REQUEST_FOR_R1_UNDER_SON1_OF_P1 - AND_son1_P2_rqR1_FOR_P1_TTCs -
AND_son1_P5_rqR1_FOR_P1_TTCs = 0

\REQUEST_FOR_R2_UNDER_SON1_OF_P1
vP1.TTCs.44: AND_son1_P2_rqR2_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 <= 0
vP1.TTCs.45: AND_son1_P2_rqR2_FOR_P1_TTCs - REQUEST_FOR_R2_WHEN_P2_IS_FATHER <= 0
vP1.TTCs.46: AND_son1_P2_rqR2_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 -
REQUEST_FOR_R2_WHEN_P2_IS_FATHER >= - 1

vP1.TTCs.47: AND_son1_P5_rqR2_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 <= 0
vP1.TTCs.48: AND_son1_P5_rqR2_FOR_P1_TTCs - REQUEST_FOR_R2_WHEN_P5_IS_FATHER <= 0
vP1.TTCs.49: AND_son1_P5_rqR2_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 -
REQUEST_FOR_R2_WHEN_P5_IS_FATHER >= - 1

vP1.TTCs.50a: AND_son1_P2_rqR2_FOR_P1_TTCs + AND_son1_P5_rqR2_FOR_P1_TTCs <= 1
vP1.TTCs.50b: REQUEST_FOR_R2_UNDER_SON1_OF_P1 - AND_son1_P2_rqR2_FOR_P1_TTCs -
AND_son1_P5_rqR2_FOR_P1_TTCs = 0

\REQUEST_FOR_R3_UNDER_SON1_OF_P1
vP1.TTCs.51: AND_son1_P2_rqR3_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 <= 0
vP1.TTCs.52: AND_son1_P2_rqR3_FOR_P1_TTCs - REQUEST_FOR_R3_WHEN_P2_IS_FATHER <= 0
vP1.TTCs.53: AND_son1_P2_rqR3_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 -
REQUEST_FOR_R3_WHEN_P2_IS_FATHER >= - 1

vP1.TTCs.54: AND_son1_P5_rqR3_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 <= 0
vP1.TTCs.55: AND_son1_P5_rqR3_FOR_P1_TTCs - REQUEST_FOR_R3_WHEN_P5_IS_FATHER <= 0
vP1.TTCs.56: AND_son1_P5_rqR3_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 -
REQUEST_FOR_R3_WHEN_P5_IS_FATHER >= - 1

vP1.TTCs.57a: AND_son1_P2_rqR3_FOR_P1_TTCs + AND_son1_P5_rqR3_FOR_P1_TTCs <= 1
vP1.TTCs.57b: REQUEST_FOR_R3_UNDER_SON1_OF_P1 - AND_son1_P2_rqR3_FOR_P1_TTCs -
AND_son1_P5_rqR3_FOR_P1_TTCs = 0
```

```
\DB_ON_R1_UNDER_SON1_OF_P1
vP1.TTCs.58: AND_son1_P2_dbR1_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 <= 0
vP1.TTCs.59: AND_son1_P2_dbR1_FOR_P1_TTCs - DB_ON_R1_WHEN_P2_IS_FATHER <= 0
vP1.TTCs.60: AND_son1_P2_dbR1_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 -
DB_ON_R1_WHEN_P2_IS_FATHER >= - 1

vP1.TTCs.61: AND_son1_P5_dbR1_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 <= 0
vP1.TTCs.62: AND_son1_P5_dbR1_FOR_P1_TTCs - DB_ON_R1_WHEN_P5_IS_FATHER <= 0
vP1.TTCs.63: AND_son1_P5_dbR1_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 -
DB_ON_R1_WHEN_P5_IS_FATHER >= - 1

vP1.TTCs.64a: AND_son1_P2_dbR1_FOR_P1_TTCs + AND_son1_P5_dbR1_FOR_P1_TTCs <= 1
vP1.TTCs.64b: DB_ON_R1_UNDER_SON1_OF_P1 - AND_son1_P2_dbR1_FOR_P1_TTCs -
AND_son1_P5_dbR1_FOR_P1_TTCs = 0

\DB_ON_R2_UNDER_SON1_OF_P1
vP1.TTCs.65: AND_son1_P2_dbR2_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 <= 0
vP1.TTCs.66: AND_son1_P2_dbR2_FOR_P1_TTCs - DB_ON_R2_WHEN_P2_IS_FATHER <= 0
vP1.TTCs.67: AND_son1_P2_dbR2_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 -
DB_ON_R2_WHEN_P2_IS_FATHER >= - 1

vP1.TTCs.68: AND_son1_P5_dbR2_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 <= 0
vP1.TTCs.69: AND_son1_P5_dbR2_FOR_P1_TTCs - DB_ON_R2_WHEN_P5_IS_FATHER <= 0
vP1.TTCs.70: AND_son1_P5_dbR2_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 -
DB_ON_R2_WHEN_P5_IS_FATHER >= - 1

vP1.TTCs.71a: AND_son1_P2_dbR2_FOR_P1_TTCs + AND_son1_P5_dbR2_FOR_P1_TTCs <= 1
vP1.TTCs.71b: DB_ON_R2_UNDER_SON1_OF_P1 - AND_son1_P2_dbR2_FOR_P1_TTCs -
AND_son1_P5_dbR2_FOR_P1_TTCs = 0

\DB_ON_R3_UNDER_SON1_OF_P1
vP1.TTCs.72: AND_son1_P2_dbR3_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 <= 0
vP1.TTCs.73: AND_son1_P2_dbR3_FOR_P1_TTCs - DB_ON_R3_WHEN_P2_IS_FATHER <= 0
vP1.TTCs.74: AND_son1_P2_dbR3_FOR_P1_TTCs - R1_RELEASED_BY_P2_FOR_P1 -
DB_ON_R3_WHEN_P2_IS_FATHER >= - 1

vP1.TTCs.75: AND_son1_P5_dbR3_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 <= 0
vP1.TTCs.76: AND_son1_P5_dbR3_FOR_P1_TTCs - DB_ON_R3_WHEN_P5_IS_FATHER <= 0
vP1.TTCs.77: AND_son1_P5_dbR3_FOR_P1_TTCs - R1_RELEASED_BY_P5_FOR_P1 -
DB_ON_R3_WHEN_P5_IS_FATHER >= - 1

vP1.TTCs.78a: AND_son1_P2_dbR3_FOR_P1_TTCs + AND_son1_P5_dbR3_FOR_P1_TTCs <= 1
vP1.TTCs.78b: DB_ON_R3_UNDER_SON1_OF_P1 - AND_son1_P2_dbR3_FOR_P1_TTCs -
AND_son1_P5_dbR3_FOR_P1_TTCs = 0
```

```
\REQUEST_FOR_R1_UNDER_SON2_OF_P1
vP1.TTCs.79: AND_son2_P4_rqR1_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 <= 0
vP1.TTCs.80: AND_son2_P4_rqR1_FOR_P1_TTCs - REQUEST_FOR_R1_WHEN_P4_IS_FATHER <= 0
vP1.TTCs.81: AND_son2_P4_rqR1_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 -
REQUEST_FOR_R1_WHEN_P4_IS_FATHER >= - 1

vP1.TTCs.82: REQUEST_FOR_R1_UNDER_SON2_OF_P1 - AND_son2_P4_rqR1_FOR_P1_TTCs = 0

\REQUEST_FOR_R2_UNDER_SON2_OF_P1
vP1.TTCs.83: AND_son2_P4_rqR2_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 <= 0
vP1.TTCs.84: AND_son2_P4_rqR2_FOR_P1_TTCs - REQUEST_FOR_R2_WHEN_P4_IS_FATHER <= 0
vP1.TTCs.85: AND_son2_P4_rqR2_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 -
REQUEST_FOR_R2_WHEN_P4_IS_FATHER >= - 1

vP1.TTCs.86: REQUEST_FOR_R2_UNDER_SON2_OF_P1 - AND_son2_P4_rqR2_FOR_P1_TTCs = 0

\REQUEST_FOR_R3_UNDER_SON2_OF_P1
vP1.TTCs.87: AND_son2_P4_rqR3_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 <= 0
vP1.TTCs.88: AND_son2_P4_rqR3_FOR_P1_TTCs - REQUEST_FOR_R3_WHEN_P4_IS_FATHER <= 0
vP1.TTCs.89: AND_son2_P4_rqR3_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 -
REQUEST_FOR_R3_WHEN_P4_IS_FATHER >= - 1

vP1.TTCs.90: REQUEST_FOR_R3_UNDER_SON2_OF_P1 - AND_son2_P4_rqR3_FOR_P1_TTCs = 0

\DB_ON_R1_UNDER_SON2_OF_P1
vP1.TTCs.91: AND_son2_P4_dbR1_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 <= 0
vP1.TTCs.92: AND_son2_P4_dbR1_FOR_P1_TTCs - DB_ON_R1_WHEN_P4_IS_FATHER <= 0
vP1.TTCs.93: AND_son2_P4_dbR1_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 -
DB_ON_R1_WHEN_P4_IS_FATHER >= - 1

vP1.TTCs.94: DB_ON_R1_UNDER_SON2_OF_P1 - AND_son2_P4_dbR1_FOR_P1_TTCs = 0

\DB_ON_R2_UNDER_SON2_OF_P1
vP1.TTCs.95: AND_son2_P4_dbR2_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 <= 0
vP1.TTCs.96: AND_son2_P4_dbR2_FOR_P1_TTCs - DB_ON_R2_WHEN_P4_IS_FATHER <= 0
vP1.TTCs.97: AND_son2_P4_dbR2_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 -
DB_ON_R2_WHEN_P4_IS_FATHER >= - 1

vP1.TTCs.98: DB_ON_R2_UNDER_SON2_OF_P1 - AND_son2_P4_dbR2_FOR_P1_TTCs = 0

\DB_ON_R3_UNDER_SON2_OF_P1
vP1.TTCs.99: AND_son2_P4_dbR3_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 <= 0
vP1.TTCs.101: AND_son2_P4_dbR3_FOR_P1_TTCs - DB_ON_R3_WHEN_P4_IS_FATHER <= 0
vP1.TTCs.102: AND_son2_P4_dbR3_FOR_P1_TTCs - R3_RELEASED_BY_P4_FOR_P1 -
DB_ON_R3_WHEN_P4_IS_FATHER >= - 1
```

```
vP1.TTCs.103: DB_ON_R3_UNDER_SON2_OF_P1 - AND_son2_P4_dbR3_FOR_P1_TTCs = 0

vP1.TTCs.104: P1_MODEL_BAD_SEQUENCE_FOR_R1 - REQUEST_FOR_R1_UNDER_SON1_OF_P1 <= 0
vP1.TTCs.105: P1_MODEL_BAD_SEQUENCE_FOR_R1 + DB_ON_R1_UNDER_SON1_OF_P1 <= 1
vP1.TTCs.106: P1_MODEL_BAD_SEQUENCE_FOR_R1 - DB_ON_R1_UNDER_SON2_OF_P1 <= 0
vP1.TTCs.107: P1_MODEL_BAD_SEQUENCE_FOR_R1 + DB_ON_R1_UNDER_SON1_OF_P1 -
REQUEST_FOR_R1_UNDER_SON1_OF_P1 - DB_ON_R1_UNDER_SON2_OF_P1 >= - 1

vP1.TTCs.108: P1_MODEL_BAD_SEQUENCE_FOR_R2 - REQUEST_FOR_R2_UNDER_SON1_OF_P1 <= 0
vP1.TTCs.109: P1_MODEL_BAD_SEQUENCE_FOR_R2 + DB_ON_R2_UNDER_SON1_OF_P1 <= 1
vP1.TTCs.110: P1_MODEL_BAD_SEQUENCE_FOR_R2 - DB_ON_R2_UNDER_SON2_OF_P1 <= 0
vP1.TTCs.111: P1_MODEL_BAD_SEQUENCE_FOR_R2 + DB_ON_R2_UNDER_SON1_OF_P1 -
REQUEST_FOR_R2_UNDER_SON1_OF_P1 - DB_ON_R2_UNDER_SON2_OF_P1 >= - 1

vP1.TTCs.112: P1_MODEL_BAD_SEQUENCE_FOR_R3 - REQUEST_FOR_R3_UNDER_SON1_OF_P1 <= 0
vP1.TTCs.113: P1_MODEL_BAD_SEQUENCE_FOR_R3 + DB_ON_R3_UNDER_SON1_OF_P1 <= 1
vP1.TTCs.114: P1_MODEL_BAD_SEQUENCE_FOR_R3 - DB_ON_R3_UNDER_SON2_OF_P1 <= 0
vP1.TTCs.115: P1_MODEL_BAD_SEQUENCE_FOR_R3 + DB_ON_R3_UNDER_SON1_OF_P1 -
REQUEST_FOR_R3_UNDER_SON1_OF_P1 - DB_ON_R3_UNDER_SON2_OF_P1 >= - 1

\P2, con accesso alle risorse: [R1:[R2]]
v18: z21 + z22 + z23 + z24 - P2_WILL_USE_R1_BY_z23 <= 1

v19: z21 + z22 + z23 <= 1
v20: z21 + z22 <= 1
v21: z21 + z23 <= 1
v22: z22 + z23 <= 1

v23: z24 - P2_WILL_USE_R1_BY_z23 = 0
v24: z21 + z24 <= 1
v25: z22 + z24 <= 1

v26: z21 + P2_WILL_USE_R1_BY_z23 <= 1
v27: z22 + P2_WILL_USE_R1_BY_z23 <= 1

v28: R1_REQUEST_MATCHED_BY_P2 - z21 - z22 - z23 = 0
v29: R2_REQUEST_MATCHED_BY_P2 - z24 = 0

v30: R2_REQUEST_LAUNCHED_BY_P2 - z21 = 0

v31: R1_MANAGED_BY_P2 - z21 >= 0
v32: R1_MANAGED_BY_P2 - z22 >= 0
v33: R1_MANAGED_BY_P2 - z23 >= 0
v34: R1_MANAGED_BY_P2 - z24 >= 0
v35: R1_MANAGED_BY_P2 - z21 - z22 - z23 - z24 <= 0
```

```
v36: R2_MANAGED_BY_P2 - z22 - z24 = 0

\ vincoli per la gestione del DB su R1 da parte di P2:
v37: R1_REQUEST_MATCHED_BY_P2 - R1_RELEASED_BY_P2_FOR_P1 <= 0

\ vincoli per la gestione del DB su R2 da parte di P2:
v38: DB_ON_R2_FOR_P2 + NO_R2_DB_FOUND_FOR_P2 - R2_REQUEST_LAUNCHED_BY_P2 = 0

v39: R2_RELEASED_BY_P3_FOR_P2 - DB_ON_R2_FOR_P2 <= 0
v40: R2_RELEASED_BY_P3_FOR_P2 - R2_REQUEST_LAUNCHED_BY_P2 <= 0
v41: R2_RELEASED_BY_P3_FOR_P2 - R2_REQUEST_MATCHED_BY_P3 <= 0
v42: R2_RELEASED_BY_P3_FOR_P2 - DB_ON_R2_FOR_P2 - R2_REQUEST_LAUNCHED_BY_P2 -
R2_REQUEST_MATCHED_BY_P3 >= - 2

v43: R2_RELEASED_BY_P4_FOR_P2 - DB_ON_R2_FOR_P2 <= 0
v44: R2_RELEASED_BY_P4_FOR_P2 - R2_REQUEST_LAUNCHED_BY_P2 <= 0
v45: R2_RELEASED_BY_P4_FOR_P2 - R2_REQUEST_MATCHED_BY_P4 <= 0
v46: R2_RELEASED_BY_P4_FOR_P2 - DB_ON_R2_FOR_P2 - R2_REQUEST_LAUNCHED_BY_P2 -
R2_REQUEST_MATCHED_BY_P4 >= - 2

v47: DB_ON_R2_FOR_P2 - R2_RELEASED_BY_P3_FOR_P2 - R2_RELEASED_BY_P4_FOR_P2 = 0

v48: R2_REQUEST_MATCHED_BY_P2 - R2_RELEASED_BY_P2_FOR_P4 <= 0


\+++ P2 TTCs +++
vP2.TTCs.1: REQUEST_FOR_R1_WHEN_P2_IS_FATHER - REQUEST_FOR_R1_UNDER_SON1_OF_P2 = 0

vP2.TTCs.2: REQUEST_FOR_R2_WHEN_P2_IS_FATHER - R2_REQUEST_LAUNCHED_BY_P2 >= 0
vP2.TTCs.3: REQUEST_FOR_R2_WHEN_P2_IS_FATHER - REQUEST_FOR_R2_UNDER_SON1_OF_P2 >= 0
vP2.TTCs.4: REQUEST_FOR_R2_WHEN_P2_IS_FATHER - R2_REQUEST_LAUNCHED_BY_P2 -
REQUEST_FOR_R2_UNDER_SON1_OF_P2 <= 0

vP2.TTCs.5: REQUEST_FOR_R3_WHEN_P2_IS_FATHER - REQUEST_FOR_R3_UNDER_SON1_OF_P2 = 0

vP2.TTCs.6: DB_ON_R1_WHEN_P2_IS_FATHER - DB_ON_R1_UNDER_SON1_OF_P2 = 0

vP2.TTCs.7: DB_ON_R2_WHEN_P2_IS_FATHER - DB_ON_R2_FOR_P2 >= 0
vP2.TTCs.8: DB_ON_R2_WHEN_P2_IS_FATHER - DB_ON_R1_UNDER_SON1_OF_P2 >= 0
vP2.TTCs.9: DB_ON_R2_WHEN_P2_IS_FATHER - DB_ON_R2_FOR_P2 - DB_ON_R1_UNDER_SON1_OF_P2
<= 0

vP2.TTCs.10: DB_ON_R3_WHEN_P2_IS_FATHER - DB_ON_R3_UNDER_SON1_OF_P2 = 0

vP2.TTCs.11: NO_SON1_FOR_P2 + DB_ON_R2_FOR_P2 = 1
```

```
\vincoli per NO_SON1_FOR_P2:
vP2.TTCs.12: REQUEST_FOR_R1_UNDER_SON1_OF_P2 + NO_SON1_FOR_P2 <= 1
vP2.TTCs.13: REQUEST_FOR_R2_UNDER_SON1_OF_P2 + NO_SON1_FOR_P2 <= 1
vP2.TTCs.14: REQUEST_FOR_R3_UNDER_SON1_OF_P2 + NO_SON1_FOR_P2 <= 1

vP2.TTCs.15: DB_ON_R1_UNDER_SON1_OF_P2 + NO_SON1_FOR_P2 <= 1
vP2.TTCs.16: DB_ON_R2_UNDER_SON1_OF_P2 + NO_SON1_FOR_P2 <= 1
vP2.TTCs.17: DB_ON_R3_UNDER_SON1_OF_P2 + NO_SON1_FOR_P2 <= 1

\REQUEST_FOR_R1_UNDER_SON1_OF_P2
vP2.TTCs.18: AND_son1_P3_rqR1_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 <= 0
vP2.TTCs.19: AND_son1_P3_rqR1_FOR_P2_TTCs - REQUEST_FOR_R1_WHEN_P3_IS_FATHER <= 0
vP2.TTCs.20: AND_son1_P3_rqR1_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 -
REQUEST_FOR_R1_WHEN_P3_IS_FATHER >= - 1

vP2.TTCs.21: AND_son1_P4_rqR1_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 <= 0
vP2.TTCs.22: AND_son1_P4_rqR1_FOR_P2_TTCs - REQUEST_FOR_R1_WHEN_P4_IS_FATHER <= 0
vP2.TTCs.23: AND_son1_P4_rqR1_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 -
REQUEST_FOR_R1_WHEN_P4_IS_FATHER >= - 1

vP2.TTCs.24a: AND_son1_P3_rqR1_FOR_P2_TTCs + AND_son1_P4_rqR1_FOR_P2_TTCs <= 1
vP2.TTCs.24b: REQUEST_FOR_R1_UNDER_SON1_OF_P2 - AND_son1_P3_rqR1_FOR_P2_TTCs -
AND_son1_P4_rqR1_FOR_P2_TTCs = 0

\REQUEST_FOR_R2_UNDER_SON1_OF_P2
vP2.TTCs.25: AND_son1_P3_rqR2_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 <= 0
vP2.TTCs.26: AND_son1_P3_rqR2_FOR_P2_TTCs - REQUEST_FOR_R2_WHEN_P3_IS_FATHER <= 0
vP2.TTCs.27: AND_son1_P3_rqR2_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 -
REQUEST_FOR_R2_WHEN_P3_IS_FATHER >= - 1

vP2.TTCs.28: AND_son1_P4_rqR2_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 <= 0
vP2.TTCs.29: AND_son1_P4_rqR2_FOR_P2_TTCs - REQUEST_FOR_R2_WHEN_P4_IS_FATHER <= 0
vP2.TTCs.30: AND_son1_P4_rqR2_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 -
REQUEST_FOR_R2_WHEN_P4_IS_FATHER >= - 1

vP2.TTCs.31a: AND_son1_P3_rqR2_FOR_P2_TTCs + AND_son1_P4_rqR2_FOR_P2_TTCs <= 1
vP2.TTCs.31b: REQUEST_FOR_R2_UNDER_SON1_OF_P2 - AND_son1_P3_rqR2_FOR_P2_TTCs -
AND_son1_P4_rqR2_FOR_P2_TTCs = 0

\REQUEST_FOR_R3_UNDER_SON1_OF_P2
vP2.TTCs.32: AND_son1_P3_rqR3_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 <= 0
vP2.TTCs.33: AND_son1_P3_rqR3_FOR_P2_TTCs - REQUEST_FOR_R3_WHEN_P3_IS_FATHER <= 0
vP2.TTCs.34: AND_son1_P3_rqR3_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 -
REQUEST_FOR_R3_WHEN_P3_IS_FATHER >= - 1

vP2.TTCs.35: AND_son1_P4_rqR3_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 <= 0
```

```
vP2.TTCs.36: AND_son1_P4_rqR3_FOR_P2_TTCs - REQUEST_FOR_R3_WHEN_P4_IS_FATHER <= 0
vP2.TTCs.37: AND_son1_P4_rqR3_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 -
REQUEST_FOR_R3_WHEN_P4_IS_FATHER >= - 1

vP2.TTCs.38a: AND_son1_P3_rqR3_FOR_P2_TTCs + AND_son1_P4_rqR3_FOR_P2_TTCs <= 1
vP2.TTCs.38b: REQUEST_FOR_R3_UNDER_SON1_OF_P2 - AND_son1_P3_rqR3_FOR_P2_TTCs -
AND_son1_P4_rqR3_FOR_P2_TTCs = 0

\DB_ON_R1_UNDER_SON1_OF_P2
vP2.TTCs.39: AND_son1_P3_dbR1_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 <= 0
vP2.TTCs.40: AND_son1_P3_dbR1_FOR_P2_TTCs - DB_ON_R1_WHEN_P3_IS_FATHER <= 0
vP2.TTCs.41: AND_son1_P3_dbR1_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 -
DB_ON_R1_WHEN_P3_IS_FATHER >= - 1

vP2.TTCs.42: AND_son1_P4_dbR1_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 <= 0
vP2.TTCs.43: AND_son1_P4_dbR1_FOR_P2_TTCs - DB_ON_R1_WHEN_P4_IS_FATHER <= 0
vP2.TTCs.44: AND_son1_P4_dbR1_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 -
DB_ON_R1_WHEN_P4_IS_FATHER >= - 1

vP2.TTCs.45a: AND_son1_P3_dbR1_FOR_P2_TTCs + AND_son1_P4_dbR1_FOR_P2_TTCs <= 1
vP2.TTCs.45b: DB_ON_R1_UNDER_SON1_OF_P2 - AND_son1_P3_dbR1_FOR_P2_TTCs -
AND_son1_P4_dbR1_FOR_P2_TTCs = 0

\DB_ON_R2_UNDER_SON1_OF_P2
vP2.TTCs.46: AND_son1_P3_dbR2_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 <= 0
vP2.TTCs.47: AND_son1_P3_dbR2_FOR_P2_TTCs - DB_ON_R2_WHEN_P3_IS_FATHER <= 0
vP2.TTCs.48: AND_son1_P3_dbR2_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 -
DB_ON_R2_WHEN_P3_IS_FATHER >= - 1

vP2.TTCs.49: AND_son1_P4_dbR2_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 <= 0
vP2.TTCs.50: AND_son1_P4_dbR2_FOR_P2_TTCs - DB_ON_R2_WHEN_P4_IS_FATHER <= 0
vP2.TTCs.51: AND_son1_P4_dbR2_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 -
DB_ON_R2_WHEN_P4_IS_FATHER >= - 1

vP2.TTCs.52: AND_son1_P3_dbR2_FOR_P2_TTCs + AND_son1_P4_dbR2_FOR_P2_TTCs <= 1
vP2.TTCs.53: DB_ON_R2_UNDER_SON1_OF_P2 - AND_son1_P3_dbR2_FOR_P2_TTCs -
AND_son1_P4_dbR2_FOR_P2_TTCs = 0

\DB_ON_R3_UNDER_SON1_OF_P2
vP2.TTCs.54: AND_son1_P3_dbR3_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 <= 0
vP2.TTCs.55: AND_son1_P3_dbR3_FOR_P2_TTCs - DB_ON_R3_WHEN_P3_IS_FATHER <= 0
vP2.TTCs.56: AND_son1_P3_dbR3_FOR_P2_TTCs - R2_RELEASED_BY_P3_FOR_P2 -
DB_ON_R3_WHEN_P3_IS_FATHER >= - 1

vP2.TTCs.57: AND_son1_P4_dbR3_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 <= 0
vP2.TTCs.58: AND_son1_P4_dbR3_FOR_P2_TTCs - DB_ON_R3_WHEN_P4_IS_FATHER <= 0
```

```
vP2.TTCs.59: AND_son1_P4_dbR3_FOR_P2_TTCs - R2_RELEASED_BY_P4_FOR_P2 -
DB_ON_R3_WHEN_P4_IS_FATHER >= - 1

vP2.TTCs.60a: AND_son1_P3_dbR3_FOR_P2_TTCs + AND_son1_P4_dbR3_FOR_P2_TTCs <= 1
vP2.TTCs.60b: DB_ON_R3_UNDER_SON1_OF_P2 - AND_son1_P3_dbR3_FOR_P2_TTCs -
AND_son1_P4_dbR3_FOR_P2_TTCs = 0


\processo P3, con accesso alle risorse: [R2]
v49: z31 - R2_REQUEST_MATCHED_BY_P3 = 0
v50: z31 - R2_MANAGED_BY_P3 = 0

\ vincoli per la gestione del DB su R2 da parte di P3:
v51: R2_REQUEST_MATCHED_BY_P3 - R2_RELEASED_BY_P3_FOR_P2 - R2_RELEASED_BY_P3_FOR_P4 <=
0

\ +++ P3 TTCs +++
v.P3.TTCs.1: REQUEST_FOR_R1_WHEN_P3_IS_FATHER = 0
v.P3.TTCs.2: REQUEST_FOR_R2_WHEN_P3_IS_FATHER = 0
v.P3.TTCs.3: REQUEST_FOR_R3_WHEN_P3_IS_FATHER = 0

v.P3.TTCs.4: DB_ON_R1_WHEN_P3_IS_FATHER = 0
v.P3.TTCs.5: DB_ON_R2_WHEN_P3_IS_FATHER = 0
v.P3.TTCs.6: DB_ON_R3_WHEN_P3_IS_FATHER = 0


\processo P4, con accesso alle risorse: [R3:[R2]]
v52: z41 + z42 + z43 + z44 - P4_WILL_USE_R3_BY_z43 <= 1

v53: z41 + z42 + z43 <= 1
v54: z41 + z42 <= 1
v55: z41 + z43 <= 1
v56: z42 + z43 <= 1

v57: z44 - P4_WILL_USE_R3_BY_z43 = 0
v58: z41 + z44 <= 1
v59: z42 + z44 <= 1

v60: z41 + P4_WILL_USE_R3_BY_z43 <= 1
v61: z42 + P4_WILL_USE_R3_BY_z43 <= 1

v62: R3_REQUEST_MATCHED_BY_P4 - z41 - z42 - z43 = 0
v63: R2_REQUEST_MATCHED_BY_P4 - z44 = 0

v64: R2_REQUEST_LAUNCHED_BY_P4 - z41 = 0
```

```
v65: R3_MANAGED_BY_P4 - z41 >= 0
v66: R3_MANAGED_BY_P4 - z42 >= 0
v67: R3_MANAGED_BY_P4 - z43 >= 0
v68: R3_MANAGED_BY_P4 - z44 >= 0
v69: R3_MANAGED_BY_P4 - z41 - z42 - z43 - z44 <= 0

v70: R2_MANAGED_BY_P4 - z42 - z44 = 0


\ vincoli per la gestione del DB su R2 da parte di P4:
v71: DB_ON_R2_FOR_P4 + NO_R2_DB_FOUND_FOR_P4 - R2_REQUEST_LAUNCHED_BY_P4 = 0

v72: R2_RELEASED_BY_P2_FOR_P4 - DB_ON_R2_FOR_P4 <= 0
v73: R2_RELEASED_BY_P2_FOR_P4 - R2_REQUEST_LAUNCHED_BY_P4 <= 0
v74: R2_RELEASED_BY_P2_FOR_P4 - R2_REQUEST_MATCHED_BY_P2 <= 0
v75: R2_RELEASED_BY_P2_FOR_P4 - DB_ON_R2_FOR_P4 - R2_REQUEST_LAUNCHED_BY_P4 -
R2_REQUEST_MATCHED_BY_P2 >= - 2

v76: R2_RELEASED_BY_P3_FOR_P4 - DB_ON_R2_FOR_P4 <= 0
v77: R2_RELEASED_BY_P3_FOR_P4 - R2_REQUEST_LAUNCHED_BY_P4 <= 0
v78: R2_RELEASED_BY_P3_FOR_P4 - R2_REQUEST_MATCHED_BY_P3 <= 0
v79: R2_RELEASED_BY_P3_FOR_P4 - DB_ON_R2_FOR_P4 - R2_REQUEST_LAUNCHED_BY_P4 -
R2_REQUEST_MATCHED_BY_P3 >= - 2

v80: DB_ON_R2_FOR_P4 - R2_RELEASED_BY_P2_FOR_P4 - R2_RELEASED_BY_P3_FOR_P4 = 0

v81: R2_REQUEST_MATCHED_BY_P4 - R2_RELEASED_BY_P4_FOR_P2 <= 0

\ vincoli per la gestione del DB su R3 da parte di P4:
v82: R3_REQUEST_MATCHED_BY_P4 - R3_RELEASED_BY_P4_FOR_P1 <= 0

\ +++ P4 TTCs +++
vP4.TTCs.1: REQUEST_FOR_R1_WHEN_P4_IS_FATHER - REQUEST_FOR_R1_UNDER_SON1_OF_P4 = 0

vP4.TTCs.2: REQUEST_FOR_R2_WHEN_P4_IS_FATHER - R2_REQUEST_LAUNCHED_BY_P4 >= 0
vP4.TTCs.3: REQUEST_FOR_R2_WHEN_P4_IS_FATHER - REQUEST_FOR_R2_UNDER_SON1_OF_P4 >= 0
vP4.TTCs.4: REQUEST_FOR_R2_WHEN_P4_IS_FATHER - R2_REQUEST_LAUNCHED_BY_P4 -
REQUEST_FOR_R2_UNDER_SON1_OF_P4 <= 0

vP4.TTCs.5: REQUEST_FOR_R3_WHEN_P4_IS_FATHER - REQUEST_FOR_R3_UNDER_SON1_OF_P4 = 0

vP4.TTCs.6: DB_ON_R1_WHEN_P4_IS_FATHER - DB_ON_R1_UNDER_SON1_OF_P4 = 0

vP4.TTCs.7: DB_ON_R2_WHEN_P4_IS_FATHER - DB_ON_R2_FOR_P4 >= 0
vP4.TTCs.8: DB_ON_R2_WHEN_P4_IS_FATHER - DB_ON_R2_UNDER_SON1_OF_P4 >= 0
```

```
vP4.TTCs.9: DB_ON_R2_WHEN_P4_IS_FATHER - DB_ON_R2_FOR_P4 - DB_ON_R2_UNDER_SON1_OF_P4
<= 0

vP4.TTCs.10: DB_ON_R3_WHEN_P4_IS_FATHER - DB_ON_R3_UNDER_SON1_OF_P4 = 0

vP4.TTCs.11: NO_SON1_FOR_P4 + DB_ON_R2_FOR_P4 = 1

\vincoli per NO_SON1_FOR_P4:
vP4.TTCs.12: REQUEST_FOR_R1_UNDER_SON1_OF_P4 + NO_SON1_FOR_P4 <= 1
vP4.TTCs.13: REQUEST_FOR_R2_UNDER_SON1_OF_P4 + NO_SON1_FOR_P4 <= 1
vP4.TTCs.14: REQUEST_FOR_R3_UNDER_SON1_OF_P4 + NO_SON1_FOR_P4 <= 1

vP4.TTCs.15: DB_ON_R1_UNDER_SON1_OF_P4 + NO_SON1_FOR_P4 <= 1
vP4.TTCs.16: DB_ON_R2_UNDER_SON1_OF_P4 + NO_SON1_FOR_P4 <= 1
vP4.TTCs.17: DB_ON_R3_UNDER_SON1_OF_P4 + NO_SON1_FOR_P4 <= 1

\REQUEST_FOR_R1_UNDER_SON1_OF_P4
vP4.TTCs.18: AND_son1_P2_rqR1_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 <= 0
vP4.TTCs.19: AND_son1_P2_rqR1_FOR_P4_TTCs - REQUEST_FOR_R1_WHEN_P2_IS_FATHER <= 0
vP4.TTCs.20: AND_son1_P2_rqR1_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 -
REQUEST_FOR_R1_WHEN_P2_IS_FATHER >= - 1

vP4.TTCs.21: AND_son1_P3_rqR1_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 <= 0
vP4.TTCs.22: AND_son1_P3_rqR1_FOR_P4_TTCs - REQUEST_FOR_R1_WHEN_P3_IS_FATHER <= 0
vP4.TTCs.23: AND_son1_P3_rqR1_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 -
REQUEST_FOR_R1_WHEN_P3_IS_FATHER >= - 1

vP4.TTCs.24a: AND_son1_P2_rqR1_FOR_P4_TTCs + AND_son1_P3_rqR1_FOR_P4_TTCs <= 1
vP4.TTCs.24b: REQUEST_FOR_R1_UNDER_SON1_OF_P4 - AND_son1_P2_rqR1_FOR_P4_TTCs -
AND_son1_P3_rqR1_FOR_P4_TTCs = 0

\REQUEST_FOR_R2_UNDER_SON1_OF_P4
vP4.TTCs.25: AND_son1_P2_rqR2_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 <= 0
vP4.TTCs.26: AND_son1_P2_rqR2_FOR_P4_TTCs - REQUEST_FOR_R2_WHEN_P2_IS_FATHER <= 0
vP4.TTCs.27: AND_son1_P2_rqR2_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 -
REQUEST_FOR_R2_WHEN_P2_IS_FATHER >= - 1

vP4.TTCs.28: AND_son1_P3_rqR2_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 <= 0
vP4.TTCs.29: AND_son1_P3_rqR2_FOR_P4_TTCs - REQUEST_FOR_R2_WHEN_P3_IS_FATHER <= 0
vP4.TTCs.30: AND_son1_P3_rqR2_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 -
REQUEST_FOR_R2_WHEN_P3_IS_FATHER >= - 1

vP4.TTCs.31a: AND_son1_P2_rqR2_FOR_P4_TTCs + AND_son1_P3_rqR2_FOR_P4_TTCs <= 1
vP4.TTCs.31b: REQUEST_FOR_R2_UNDER_SON1_OF_P4 - AND_son1_P2_rqR2_FOR_P4_TTCs -
AND_son1_P3_rqR2_FOR_P4_TTCs = 0
```

```
\REQUEST_FOR_R3_UNDER_SON1_OF_P4
vP4.TTCs.32: AND_son1_P2_rqR3_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 <= 0
vP4.TTCs.33: AND_son1_P2_rqR3_FOR_P4_TTCs - REQUEST_FOR_R3_WHEN_P2_IS_FATHER <= 0
vP4.TTCs.34: AND_son1_P2_rqR3_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 -
REQUEST_FOR_R3_WHEN_P2_IS_FATHER >= - 1

vP4.TTCs.35: AND_son1_P3_rqR3_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 <= 0
vP4.TTCs.36: AND_son1_P3_rqR3_FOR_P4_TTCs - REQUEST_FOR_R3_WHEN_P3_IS_FATHER <= 0
vP4.TTCs.37: AND_son1_P3_rqR3_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 -
REQUEST_FOR_R3_WHEN_P3_IS_FATHER >= - 1

vP4.TTCs.38a: AND_son1_P2_rqR3_FOR_P4_TTCs + AND_son1_P3_rqR3_FOR_P4_TTCs <= 1
vP4.TTCs.38b: REQUEST_FOR_R3_UNDER_SON1_OF_P4 - AND_son1_P2_rqR3_FOR_P4_TTCs -
AND_son1_P3_rqR3_FOR_P4_TTCs = 0

\DB_ON_R1_UNDER_SON1_OF_P4
vP4.TTCs.39: AND_son1_P2_dbR1_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 <= 0
vP4.TTCs.40: AND_son1_P2_dbR1_FOR_P4_TTCs - DB_ON_R1_WHEN_P2_IS_FATHER <= 0
vP4.TTCs.41: AND_son1_P2_dbR1_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 -
DB_ON_R1_WHEN_P2_IS_FATHER >= - 1

vP4.TTCs.42: AND_son1_P3_dbR1_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 <= 0
vP4.TTCs.43: AND_son1_P3_dbR1_FOR_P4_TTCs - DB_ON_R1_WHEN_P3_IS_FATHER <= 0
vP4.TTCs.44: AND_son1_P3_dbR1_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 -
DB_ON_R1_WHEN_P3_IS_FATHER >= - 1

vP4.TTCs.45a: AND_son1_P2_dbR1_FOR_P4_TTCs + AND_son1_P3_dbR1_FOR_P4_TTCs <= 1
vP4.TTCs.45b: DB_ON_R1_UNDER_SON1_OF_P4 - AND_son1_P2_dbR1_FOR_P4_TTCs -
AND_son1_P3_dbR1_FOR_P4_TTCs = 0

\DB_ON_R2_UNDER_SON1_OF_P4
vP4.TTCs.46: AND_son1_P2_dbR2_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 <= 0
vP4.TTCs.47: AND_son1_P2_dbR2_FOR_P4_TTCs - DB_ON_R2_WHEN_P2_IS_FATHER <= 0
vP4.TTCs.48: AND_son1_P2_dbR2_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 -
DB_ON_R2_WHEN_P2_IS_FATHER >= - 1

vP4.TTCs.49: AND_son1_P3_dbR2_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 <= 0
vP4.TTCs.50: AND_son1_P3_dbR2_FOR_P4_TTCs - DB_ON_R2_WHEN_P3_IS_FATHER <= 0
vP4.TTCs.51: AND_son1_P3_dbR2_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 -
DB_ON_R2_WHEN_P3_IS_FATHER >= - 1

vP4.TTCs.52: AND_son1_P2_dbR2_FOR_P4_TTCs + AND_son1_P3_dbR2_FOR_P4_TTCs <= 1
vP4.TTCs.53: DB_ON_R2_UNDER_SON1_OF_P4 - AND_son1_P2_dbR2_FOR_P4_TTCs -
AND_son1_P3_dbR2_FOR_P4_TTCs = 0

\DB_ON_R3_UNDER_SON1_OF_P4
```

```
vP4.TTCs.54: AND_son1_P2_dbR3_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 <= 0
vP4.TTCs.55: AND_son1_P2_dbR3_FOR_P4_TTCs - DB_ON_R3_WHEN_P2_IS_FATHER <= 0
vP4.TTCs.56: AND_son1_P2_dbR3_FOR_P4_TTCs - R2_RELEASED_BY_P2_FOR_P4 -
DB_ON_R3_WHEN_P2_IS_FATHER >= - 1

vP4.TTCs.57: AND_son1_P3_dbR3_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 <= 0
vP4.TTCs.58: AND_son1_P3_dbR3_FOR_P4_TTCs - DB_ON_R3_WHEN_P3_IS_FATHER <= 0
vP4.TTCs.59: AND_son1_P3_dbR3_FOR_P4_TTCs - R2_RELEASED_BY_P3_FOR_P4 -
DB_ON_R3_WHEN_P3_IS_FATHER >= - 1

vP4.TTCs.60a: AND_son1_P2_dbR3_FOR_P4_TTCs + AND_son1_P3_dbR3_FOR_P4_TTCs <= 1
vP4.TTCs.60b: DB_ON_R3_UNDER_SON1_OF_P4 - AND_son1_P2_dbR3_FOR_P4_TTCs -
AND_son1_P3_dbR3_FOR_P4_TTCs = 0

\processo P5 (min pr), con accesso alle risorse: [R1]
v83: z51 - R1_REQUEST_MATCHED_BY_P5 = 0
v84: z51 - R1_MANAGED_BY_P5 = 0

\ vincoli per la gestione del DB su R1 da parte di P4:
v85: R1_REQUEST_MATCHED_BY_P5 - R1_RELEASED_BY_P5_FOR_P1 <= 0

\ +++ P5 TTCs +++
v.P5.TTCs.1: REQUEST_FOR_R1_WHEN_P5_IS_FATHER = 0
v.P5.TTCs.2: REQUEST_FOR_R2_WHEN_P5_IS_FATHER = 0
v.P5.TTCs.3: REQUEST_FOR_R3_WHEN_P5_IS_FATHER = 0

v.P5.TTCs.4: DB_ON_R1_WHEN_P5_IS_FATHER = 0
v.P5.TTCs.5: DB_ON_R2_WHEN_P5_IS_FATHER = 0
v.P5.TTCs.6: DB_ON_R3_WHEN_P5_IS_FATHER = 0
\ +++ FINE +++


\vincoli per la gestione della risorsa R1
v86: R1_REQUEST_MATCHED_BY_P2 + R1_REQUEST_MATCHED_BY_P5 <= 1
v87: OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY - R1_REQUEST_MATCHED_BY_P2 -
R1_REQUEST_MATCHED_BY_P5 = 0
v88: DB_ON_R1_FOR_P1 - OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY <= 0

v89: OR_RESULT_FOR_DB_ON_R1_PROPERTY - DB_ON_R1_FOR_P1 = 0
v90: R1_RELEASED_BY_P2_FOR_P1 + R1_RELEASED_BY_P5_FOR_P1 -
OR_RESULT_FOR_DB_ON_R1_PROPERTY <= 0

\vincoli per la gestione della risorsa R2
v91: R2_REQUEST_MATCHED_BY_P2 + R2_REQUEST_MATCHED_BY_P3 + R2_REQUEST_MATCHED_BY_P4 <=
1
```

```
v92: OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY - R2_REQUEST_MATCHED_BY_P2 -
R2_REQUEST_MATCHED_BY_P3 - R2_REQUEST_MATCHED_BY_P4 = 0
v93: DB_ON_R2_FOR_P2 + DB_ON_R2_FOR_P4 - OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY <=
0
\condizione legata ad un OR_RESULT := {ESISTE UN Pj che MATCHED Rj}.

v94: OR_RESULT_FOR_DB_ON_R2_PROPERTY - DB_ON_R2_FOR_P2 - DB_ON_R2_FOR_P4 = 0
v95: R2_RELEASED_BY_P3_FOR_P2 + R2_RELEASED_BY_P3_FOR_P4 + R2_RELEASED_BY_P2_FOR_P4 +
R2_RELEASED_BY_P4_FOR_P2 - OR_RESULT_FOR_DB_ON_R2_PROPERTY <= 0 \CORRETTO
\introduzione della mutua esclusione tra le variabili che possono liberare una data
risorsa Rj e QUI UN OR LEGATO AI ESISTE UN DB SU RJ

\vincoli per la gestione della risorsa R3
v96: R3_REQUEST_MATCHED_BY_P4 <= 1
v97: OR_RESULT_FOR_R3_REQUEST_MATCHED_PROPERTY - R3_REQUEST_MATCHED_BY_P4 = 0
v98: DB_ON_R3_FOR_P1 - OR_RESULT_FOR_R3_REQUEST_MATCHED_PROPERTY <= 0

v99: OR_RESULT_FOR_DB_ON_R3_PROPERTY - DB_ON_R3_FOR_P1 = 0
v100: R3_RELEASED_BY_P4_FOR_P1 - OR_RESULT_FOR_DB_ON_R3_PROPERTY <= 0

\vincoli per {R1_MANAGED_BY_P2,5}
v101: R1_MANAGED_BY_P2 + R1_MANAGED_BY_P5 <= 1

\vincoli per {R2_MANAGED_BY_P2,3,4}
v103: R2_MANAGED_BY_P2 + R2_MANAGED_BY_P3 + R2_MANAGED_BY_P4 <= 1

Bounds

Binary

 \P1 VARs
 R1_REQUEST_LAUNCHED_BY_P1
 DB_ON_R1_FOR_P1
 NO_R1_DB_FOUND_FOR_P1
 R3_REQUEST_LAUNCHED_BY_P1
 DB_ON_R3_FOR_P1
 NO_R3_DB_FOUND_FOR_P1

       \P1 TTCs VARs
       REQUEST_FOR_R1_WHEN_P1_IS_FATHER
       REQUEST_FOR_R2_WHEN_P1_IS_FATHER
       REQUEST_FOR_R3_WHEN_P1_IS_FATHER

       DB_ON_R1_WHEN_P1_IS_FATHER
       DB_ON_R2_WHEN_P1_IS_FATHER
       DB_ON_R3_WHEN_P1_IS_FATHER
```

```
NO_SON1_FOR_P1
NO_SON2_FOR_P1

\SON1_OF_P1
REQUEST_FOR_R1_UNDER_SON1_OF_P1
REQUEST_FOR_R2_UNDER_SON1_OF_P1
REQUEST_FOR_R3_UNDER_SON1_OF_P1

DB_ON_R1_UNDER_SON1_OF_P1
DB_ON_R2_UNDER_SON1_OF_P1
DB_ON_R3_UNDER_SON1_OF_P1

\SON2_OF_P1
REQUEST_FOR_R1_UNDER_SON2_OF_P1
REQUEST_FOR_R2_UNDER_SON2_OF_P1
REQUEST_FOR_R3_UNDER_SON2_OF_P1

DB_ON_R1_UNDER_SON2_OF_P1
DB_ON_R2_UNDER_SON2_OF_P1
DB_ON_R3_UNDER_SON2_OF_P1

\AS SON1
AND_son1_P2_rqR1_FOR_P1_TTCs
AND_son1_P2_rqR2_FOR_P1_TTCs
AND_son1_P2_rqR3_FOR_P1_TTCs

AND_son1_P2_dbR1_FOR_P1_TTCs
AND_son1_P2_dbR2_FOR_P1_TTCs
AND_son1_P2_dbR3_FOR_P1_TTCs

AND_son1_P5_rqR1_FOR_P1_TTCs
AND_son1_P5_rqR2_FOR_P1_TTCs
AND_son1_P5_rqR3_FOR_P1_TTCs
AND_son1_P5_dbR1_FOR_P1_TTCs
AND_son1_P5_dbR2_FOR_P1_TTCs
AND_son1_P5_dbR3_FOR_P1_TTCs

\AS SON2
AND_son2_P4_rqR1_FOR_P1_TTCs
AND_son2_P4_rqR2_FOR_P1_TTCs
AND_son2_P4_rqR3_FOR_P1_TTCs

AND_son2_P4_dbR1_FOR_P1_TTCs
AND_son2_P4_dbR2_FOR_P1_TTCs
AND_son2_P4_dbR3_FOR_P1_TTCs
```

```
\PENALTY VARs OF P1
P1_MODEL_BAD_SEQUENCE_FOR_R1
P1_MODEL_BAD_SEQUENCE_FOR_R2
P1_MODEL_BAD_SEQUENCE_FOR_R3

\P2 VARs
z21
z22
z23
z24

R1_MANAGED_BY_P2
R1_REQUEST_MATCHED_BY_P2
P2_WILL_USE_R1_BY_z23

R2_REQUEST_LAUNCHED_BY_P2
R2_MANAGED_BY_P2
R2_REQUEST_MATCHED_BY_P2

DB_ON_R2_FOR_P2
NO_R2_DB_FOUND_FOR_P2

R2_RELEASED_BY_P2_FOR_P4
R1_RELEASED_BY_P2_FOR_P1

      \P2 TTCs VARs
      REQUEST_FOR_R1_WHEN_P2_IS_FATHER
      REQUEST_FOR_R2_WHEN_P2_IS_FATHER
      REQUEST_FOR_R3_WHEN_P2_IS_FATHER

      DB_ON_R1_WHEN_P2_IS_FATHER
      DB_ON_R2_WHEN_P2_IS_FATHER
      DB_ON_R3_WHEN_P2_IS_FATHER
      NO_SON1_FOR_P2

      REQUEST_FOR_R1_UNDER_SON1_OF_P2
      REQUEST_FOR_R2_UNDER_SON1_OF_P2
      REQUEST_FOR_R3_UNDER_SON1_OF_P2

      DB_ON_R1_UNDER_SON1_OF_P2
      DB_ON_R2_UNDER_SON1_OF_P2
      DB_ON_R3_UNDER_SON1_OF_P2

      AND_son1_P3_rqR1_FOR_P2_TTCs
      AND_son1_P3_rqR2_FOR_P2_TTCs
```

```
      AND_son1_P3_rqR3_FOR_P2_TTCs

      AND_son1_P4_rqR1_FOR_P2_TTCs
      AND_son1_P4_rqR2_FOR_P2_TTCs
      AND_son1_P4_rqR3_FOR_P2_TTCs

      AND_son1_P3_dbR1_FOR_P2_TTCs
      AND_son1_P3_dbR2_FOR_P2_TTCs
      AND_son1_P3_dbR3_FOR_P2_TTCs

      AND_son1_P4_dbR1_FOR_P2_TTCs
      AND_son1_P4_dbR2_FOR_P2_TTCs
      AND_son1_P4_dbR3_FOR_P2_TTCs

\P3 VARs
 z31
 R2_REQUEST_MATCHED_BY_P3
 R2_MANAGED_BY_P3

 R2_RELEASED_BY_P3_FOR_P2
 R2_RELEASED_BY_P3_FOR_P4

      \P3 TTCs VARs
      REQUEST_FOR_R1_WHEN_P3_IS_FATHER
      REQUEST_FOR_R2_WHEN_P3_IS_FATHER
      REQUEST_FOR_R3_WHEN_P3_IS_FATHER

      DB_ON_R1_WHEN_P3_IS_FATHER
      DB_ON_R2_WHEN_P3_IS_FATHER
      DB_ON_R3_WHEN_P3_IS_FATHER

\P4 VARs
 z41
 z42
 z43
 z44

 R3_MANAGED_BY_P4
 R3_REQUEST_MATCHED_BY_P4
 P4_WILL_USE_R3_BY_z43

 R2_REQUEST_LAUNCHED_BY_P4
 R2_MANAGED_BY_P4
 R2_REQUEST_MATCHED_BY_P4
```

```
DB_ON_R2_FOR_P4
NO_R2_DB_FOUND_FOR_P4

R2_RELEASED_BY_P4_FOR_P2

R3_RELEASED_BY_P4_FOR_P1

     \P4 TTCs VARs
     REQUEST_FOR_R1_WHEN_P4_IS_FATHER
     REQUEST_FOR_R2_WHEN_P4_IS_FATHER
     REQUEST_FOR_R3_WHEN_P4_IS_FATHER

     DB_ON_R1_WHEN_P4_IS_FATHER
     DB_ON_R2_WHEN_P4_IS_FATHER
     DB_ON_R3_WHEN_P4_IS_FATHER

     NO_SON1_FOR_P4

     REQUEST_FOR_R1_UNDER_SON1_OF_P4
     REQUEST_FOR_R2_UNDER_SON1_OF_P4
     REQUEST_FOR_R3_UNDER_SON1_OF_P4

     DB_ON_R1_UNDER_SON1_OF_P4
     DB_ON_R2_UNDER_SON1_OF_P4
     DB_ON_R3_UNDER_SON1_OF_P4

     AND_son1_P2_rqR1_FOR_P4_TTCs
     AND_son1_P2_rqR2_FOR_P4_TTCs
     AND_son1_P2_rqR3_FOR_P4_TTCs

     AND_son1_P3_rqR1_FOR_P4_TTCs
     AND_son1_P3_rqR2_FOR_P4_TTCs
     AND_son1_P3_rqR3_FOR_P4_TTCs

     AND_son1_P2_dbR1_FOR_P4_TTCs
     AND_son1_P2_dbR2_FOR_P4_TTCs
     AND_son1_P2_dbR3_FOR_P4_TTCs

     AND_son1_P3_dbR1_FOR_P4_TTCs
     AND_son1_P3_dbR2_FOR_P4_TTCs
     AND_son1_P3_dbR3_FOR_P4_TTCs

\P5 VARs
z51
R1_REQUEST_MATCHED_BY_P5
R1_MANAGED_BY_P5
```

```
 R1_RELEASED_BY_P5_FOR_P1

       \P5 TTCs VARs
       REQUEST_FOR_R1_WHEN_P5_IS_FATHER
       REQUEST_FOR_R2_WHEN_P5_IS_FATHER
       REQUEST_FOR_R3_WHEN_P5_IS_FATHER

       DB_ON_R1_WHEN_P5_IS_FATHER
       DB_ON_R2_WHEN_P5_IS_FATHER
       DB_ON_R3_WHEN_P5_IS_FATHER

 \R1 VARs
 OR_RESULT_FOR_R1_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R1_PROPERTY

 \R2
 OR_RESULT_FOR_R2_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R2_PROPERTY

 \R3 VARs
 OR_RESULT_FOR_R3_REQUEST_MATCHED_PROPERTY
 OR_RESULT_FOR_DB_ON_R3_PROPERTY

End
```

CONCLUSIONI

Meriti principali del lavoro svolto

La prima conclusione circa il lavoro svolto riguarda la facilità con cui è possibile *adoperare le regole di modellazione* illustrate, agli scenari costituiti da *una vasta tipologia di applicazioni r-t*. Ciò è giustificato dal fatto che: si è scelto di modellare *i comportamenti dei singoli task*, e non considerarli come semplici insiemi di sezioni critiche. Questo rende possibile la rappresentazione, in termini di vincoli lineari, di processi *con le più differenziate modalità di accesso alle risorse condivise*.

Tale idea è stata adottata anche nell'ambito della rappresentazione delle *interazioni bloccanti tra processi.* In particolare, prescindendo essa dal far diretto riferimento alle singole sezioni critiche dei task, ed essendo invece tali interazioni descritte adoperando le *proprietà di comportamento* dei processi, allora è consentito considerare questo come un ulteriore *elemento portabile* delle tecniche di modellazione descritte.

Un altro pregio attribuibile a questa strategia di rappresentazione è quella di riuscire a tener traccia dei possibili ordini con cui gli elementi del Resource Set possano essere acceduti dai task che costituiscono l'applicazione r-t, quindi di fornire informazioni circa *la precisa sequenza di esecuzione dei processi che massimizzano il tempo blocco del processo più critico*.

La soluzione ottima offre infatti precise indicazioni riguardo *il caso peggiore di schedulazione RMPO dei processi dell'applicazione in esame, che determina il valore massimo per il parametro* BP_1.

Sempre in questo ambito, è notevole come la descrizione dell'ordine degli accessi avvenga in termini di *strutture ad albero ad un livello* per ogni processo, i quali sono interconnessi vicendevolmente grazie ad opportune *variabili di interfaccia*, che rendono disponibili per ciascun nodo radice di tali strutture, informazioni circa eventi che sono associati a tutti i relativi nodi figli.

Con questo metodo dunque non è necessario *costruire l'intero albero* che *globalmente* tenga conto delle possibili *sequenze di blocchi ammissibili da un punto di vista temporale*, ma semplicemente alberi elementari da interconnettere tra loro ancora una volta sfruttando le proprietà delle interazioni bloccanti tra task.

Infine, l'ultimo merito di questo elaborato riguarda il fatto che attraverso la metodologia di modellazione ILP descritta nei capitoli procedenti, è effettivamente possibile risolvere il problema della determinazione del massimo tempo di blocco B_{P1} associato al processo più critico, *in un tempo irrisorio* e con l'ausilio di un software di ottimizzazione che riceva in ingresso il modello ILP

dell'applicazione (in questi casi la complessità computazionale è di tipo polinomiale), cercando così di colmare i *punti deboli* della strategia di calcolo proposta da Rajkumar.

Possibili nuovi sviluppi e miglioramenti

Il presente elaborato non può tuttavia essere considerato *come esaustivo del compito* per cui è stato inizialmente ideato, e questo principalmente per una ragione: le interazioni bloccanti a cui si è dato principale risalto nella modellazione ILP definita, sono *i blocchi diretti e quelli transitivi* che possono occorrere per una data applicazione r-t.

In effetti, è facile notare come nei precedenti capitoli l'obiettivo principale fosse il calcolo del tempo di blocco del processo a *massima priorità* P_1. Ciò è giustificato dal fatto che *l'idea guida* della progettazione era quella di definire Modelli ILP che *simulassero il lancio del processo* P_i *di cui si intendeva determinare il relativo parametro* $B_{Pi,}$ quindi individuare i task di priorità inferiore che con le proprie sezioni critiche *massimizzassero* i suoi tempi di arresto.

Ciò ha condotto quindi a *focalizzarsi* specificatamente sul processo più critico di un'applicazione r-t, non considerando immediatamente l'importanza di dover rappresentare per i task a priorità inferiore anche gli eventuali *blocchi di tipo push-through*, qualora fosse stato *necessario* calcolare per essi il relativo parametro B_{Pi}. La loro modellazione, sempre in termini di *proprietà di comportamento dei singoli processi*, è lasciata quindi al Lettore di questo lavoro.

In ultimo, un elemento che *renderebbe quanto progettato* maggiormente fruibile, sarebbe l'implementazione di un software tale che: sulla base delle *regole di definizione dei vincoli lineari descritte nei precedenti capitoli*, e fornendo in ingresso una rappresentazione secondo un'opportuna *grammatica/sintassi* dell'applicazione r-t in esame, nonché l'indicazione del task di cui si vuole calcolare il massimo tempo di blocco, produca in uscita *il Modello ILP finale* da "risolvere" con l'ausilio di un proprio software di ottimizzazione.

BIBLIOGRAFIA

[1] Buttazzo C. G., *"Hard Real-Time Computing System Predictable - Scheduling Algorithms and Applications – Third Edition"*, Springer Science+Business Media LLC, New York USA, 2011.

[2] Bigi G., Cappanera P., Frangioni A., Gallo G., Pallottino S., Scaparra M. P., Scutellà M. G., Relazioni binarie, in "*Appunti di Ricerca Operativa 2015/2016"*, Dipartimento di Informatica Università degli Studi di Pisa **(**http://www.di.unipi.it/optimize/Courses), Pisa, 2015.

[3] Cavallari A., "*Un nuovo metodo per la determinazione delle prestazioni conseguibili con il protocollo "Priority Inheritance" in applicazioni multitasking Hard Real-Time"*, Università di Bologna, Bologna, 2015.

[4] DeMichelis G., *"Introduzione ai Sistemi Real Time"* pp. 103, Lulu.com, 2012.

[5] Faldella E.,
- *"Introduzione"* (http://lia.deis.unibo.it/Courses/SistRT/contenuti/2%20Materiale%20didattico/1%20Introduzione.pdf), Corso di Sistemi in Tempo Reale M, Università di Bologna, Bologna.
- *"Scheduling di processi hard real-time" (http://lia.deis.unibo.it/Courses/SistRT/contenuti/2%20Materiale%20didattico/2%20Scheduling%20di%20processi%20hard%20real-time.pdf)*, Corso di Sistemi in Tempo Reale M, Università di Bologna, Bologna.
- *"Scheduling di processi in presenza di risorse condivise" (http://lia.deis.unibo.it/Courses/SistRT/contenuti/2%20Materiale%20didattico/4%20Protocolli%20di%20accesso%20a%20risorse%20condivise.pdf)*, Corso di Sistemi in Tempo Reale M, Università di Bologna, Bologna.

[6] Piscitelli G., "*Sistemi in tempo reale"*, Politecnico di Bari, Bari, 2003.

[7] Rajkumar R, "*Synchronization in Real-Time System - A Priority Inheritance Approach"*, Dordrecht, Kluwer Academic Publishers, 1991.