UNIVERSITÀ DEGLI STUDI
DI MILANO

Alcuni commenti su Lab07

Alcuni dei compiti erano da farsi quasi in automatico... ma nella maniera giusta

Iterable

Sbagliato

```java
@NotNull
@Override
public Iterator<Card> iterator() {
  return this.cards.iterator();
}
```

```java
@NotNull
@Override
public Iterator<Card> iterator() {
  return Collections.unmodifiableCollection(cards).iterator();
}
```

```java
@NotNull
@Override
public Iterator<Card> iterator() {
  List<Card> cardcopy = new ArrayList<>(cards);
  return cardcopy.iterator();
}
```

```java
@NotNull
@Override
public Iterator<Card> iterator() {
  return List.copyOf(cards).iterator();
}
```

Comparable

Sbagliato e senza test!!

```
@Override
public int compareTo(@NotNull Player o) {
  if(this.playedCard().getRank().points()>o.playedCard().getRank().points())
    return 1;
  if((this.playedCard().getRank().points()>o.playedCard().getRank().points()))
    return -1;
  return 0;
}
```

Interfaccia sbagliata

```
public class Player implements Iterable<Card>, Comparator<Player> {

  @Override
  public int compare(Player o1, Player o2) {
    if(this.getPoints() > o1.getPoints()) return 1;
    if(this.getPoints() < o2.getPoints()) return -1;
    return 0;
  }

...
   int result= firstCardPlayer.compare(firstCardPlayer,otherPlayer(firstCardPlayer));
...
}
```

Comparable (cont)

```
public class Player implements Iterable , Comparable {
  @Override
  public int compareTo(@NotNull Object o) {
    Player p2 = (Player)o;
    return this.getPoints()- p2.getPoints();
  }
}
```

Interfaccia non generica... quindi...

```
@Test
public void comparablePlayer(){
  Card c1 = Card.get(Rank.TRE,Suit.COPPE);
  Player p1 = new Player("Giocatore_1");
  Card c2 = Card.get(Rank.ASSO,Suit.COPPE);
  Player p2 = new Player("Giocatore_2");
  p1.personalDeck.add(c1);
  p2.personalDeck.add(c2);
  assertThat(p1.compareTo(p2)).isEqualTo(-1);
}
```

Comparable

```java
@Override
public int compareTo(@NotNull Player player) {
  return this.getPoints() - player.getPoints();
}
```

```java
@Override
public int compareTo(@NotNull Player player) {
  return Integer.compare(this.getPoints(), o.getPoints());
}
```

Giusto

```java
@Override
public int compareTo(@NotNull Player player) {
  int personal_points = 0;
  for (Card c: personalDeck)
    personal_points += c.getRank().points();

  int other_points = 0;
  for (Card c: player.personalDeck)
    other_points += c.getRank().points();

  return Integer.compare(personal_points, other_points);
}
```

```java
@Override
public int compareTo(@NotNull Player player) {
  if (getPoints() == o.getPoints())
    return 0;
  else if (getPoints() < o.getPoints())
    return -1;
  return 1;
}
```

Avvertenze su Comparable prese da Effective Java

classes. **Use of the relational operators < and > in compareTo methods is verbose and error-prone and no longer recommended.**

Occasionally you may see compareTo or compare methods that rely on the fact that the difference between two values is negative if the first value is less than the second, zero if the two values are equal, and positive if the first value is greater. Here is an example:

```
// BROKEN difference-based comparator - violates transitivity!
static Comparator<Object> hashCodeOrder = new Comparator<>() {
    public int compare(Object o1, Object o2) {
        return o1.hashCode() - o2.hashCode();
    }
};
```

Do not use this technique. It is fraught with danger from int
IEEE 754 floating point arithmetic artifacts [JLS 15.20.1, 15.2
the resulting methods are unlikely to be significantly faster
using the techniques described in this item. Use either a static c

```
private static final Comparator<Player> COMPARATOR =
   comparingInt(Player::getPoints);

@Override
public int compareTo(@NotNull Player player) {
  return COMPARATOR.compare(this, player);
}
```

Null Object pattern

```java
public class StrategiaNullObj implements Strategy{
  @Override
  public @NotNull Card chooseCard(@NotNull Player me, @NotNull Player other, @NotNull Suit briscola) {
      return me.iterator().next();
  }
}
```

Sbagliato: non è un Object

```java
public interface Strategy {
  @NotNull
  Card chooseCard(@NotNull Player me, @NotNull Player other, @NotNull Suit briscola);

  public static Strategy NULL = new Strategy() {
    public Card chooseCard(@NotNull Player me, @NotNull Player other, @NotNull Suit briscola) {
      return null;
    }
  };
}
```

Sbagliato: ritorna null !

Null Object pattern

```java
public static Player NULL = new Player("null") {
    @Override
    public @NotNull String toString() { return null; }
    @Override
    public @NotNull Iterator<Card> iterator() { return null; }
    @Override
    public int compareTo(@NotNull Player o) { return -1; }
    @Override
    public @NotNull Card chooseFirstCard(@NotNull Briscola game) { return null; }
    @Override
    public @NotNull Card chooseSecondCard(@NotNull Briscola game) { return null; }
    @Override
    public Card playedCard() { return null; }
    @Override
    public @NotNull String getName() { return null; }
  };
```

Tutti questi `return null` in presenza di annotazione `@NotNull` in un pattern che dovrebbe evitare l'uso dei `null`…

… e non c'è l'unico metodo che andava ridefinito: `shout`.

Nelle classi di test

- Uso di mockito in maniera molto … *naïve*
- Pochi test e mancanza della copertura
  - comparable si guarda solo se risponde 1 ad un caso…
    - funzionerebbe anche con `return 1;`

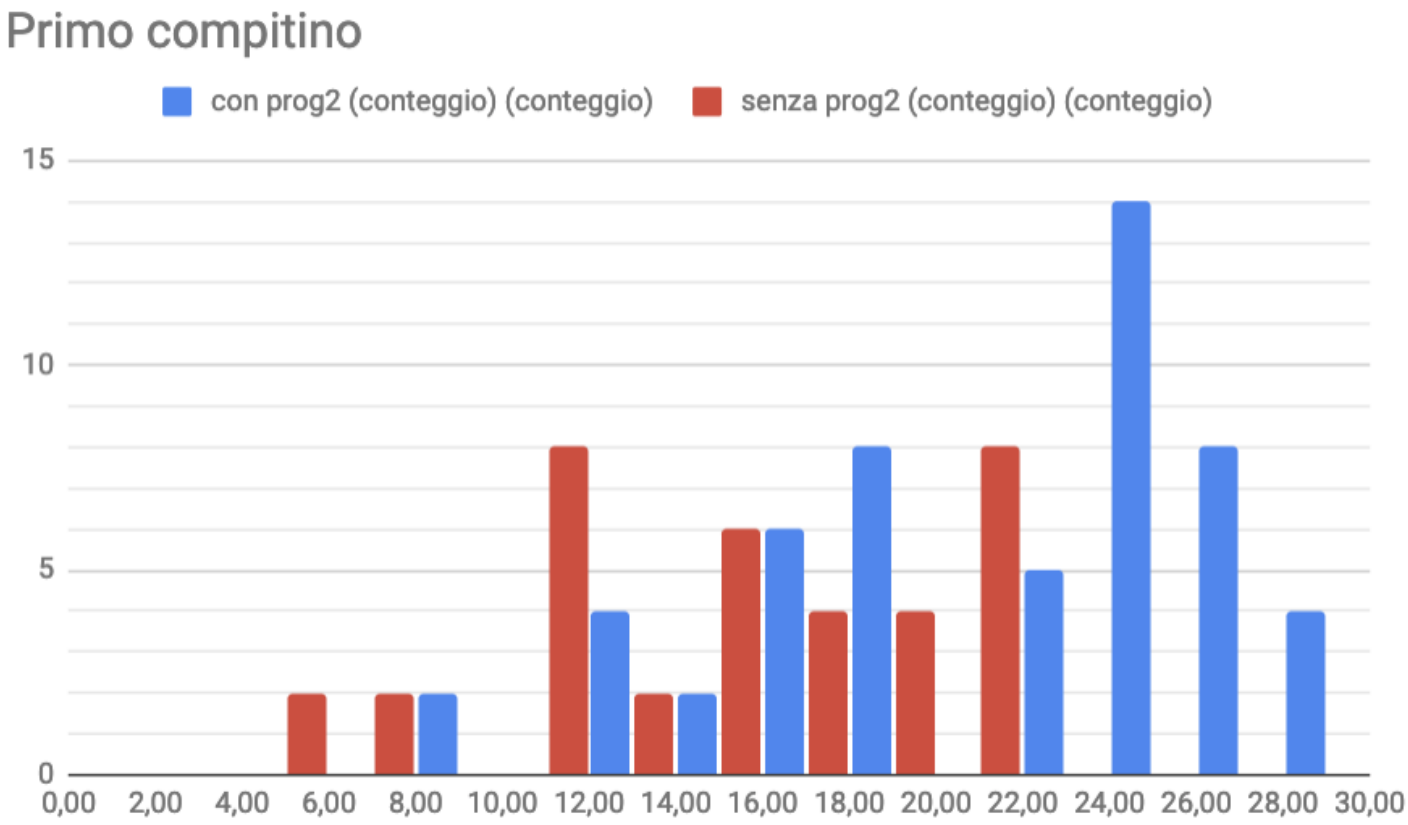
```
@Test void TestIterator(){
  Player SUT= new Player("Carlo");
  List<Card> mano= List.of( new Card[]{
      Card.get(Rank.ASSO, Suit.BASTONI),
      Card.get(Rank.TRE,Suit.BASTONI),
      Card.get(Rank.CINQUE,Suit.BASTONI)});
  for (Card c: mano)
      SUT.giveCard(c);
  for (Card c: SUT)
      assertThat(mano.contains(c)).isTrue();
}
```

Sbagliato: le asserzioni potrebbero non essere neanche eseguite per diversi tipi di errori !

Strategie

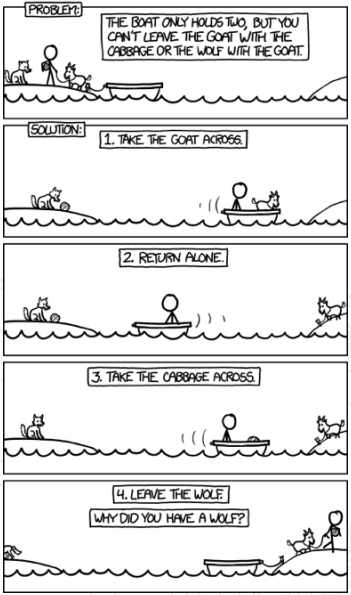
- Diversi non ci sono neanche arrivati o quasi
- Quasi nessuno ne ha fatte 3+3
- Quasi nessuno ha fatto ragionamenti "sensati" o almeno che guardassero le carte dell'avversario o la carta che aveva giocato
- Diversi non hanno capito il Chain Of Responsability pattern

Voti


Primo compitino
con prog2 (conteggio) (conteggio)
senza prog2 (conteggio) (conteggio)
15
10
5
0
0,00
2,00
4,00
6,00
8,00
10,00
12,00
14,00
16,00
18,00
20,00
22,00
24,00
26,00
28,00
30,00

Esercizio

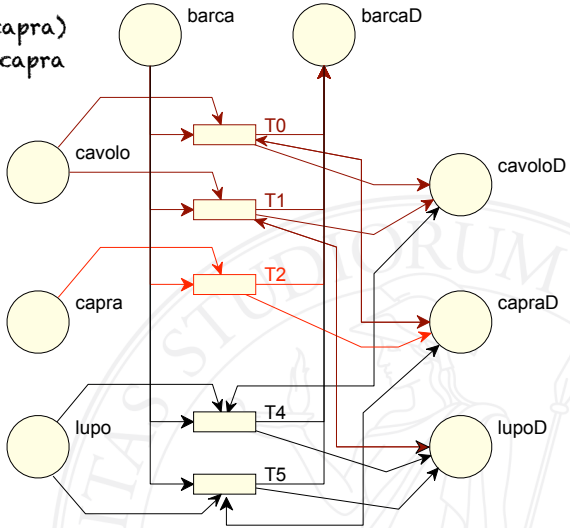
- Modellare il problema del barcaiolo che deve traghettare da una sponda all'altra di un torrente un lupo, una capra e un cavolo con una barca di capacità 1 (può trasportare solo un elemento alla volta oltre al barcaiolo).
- Il trasporto è vincolato dalla necessità di non lasciare soli
- lupo e capra
- capra e cavolo



http://xkcd.com/1134/

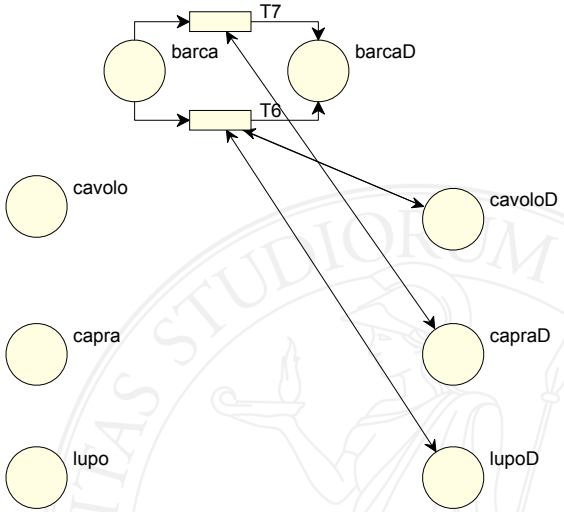
Soluzione

Lupo: NOT(cavolo AND capra)
== NOT cavolo OR NOT capra

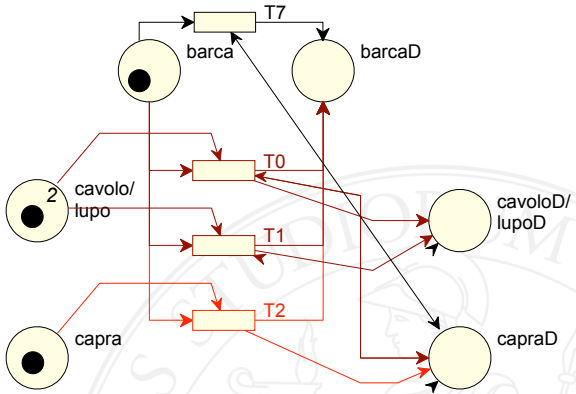


Soluzione (cont.)

Barcaiolo da solo



Soluzione alternativa

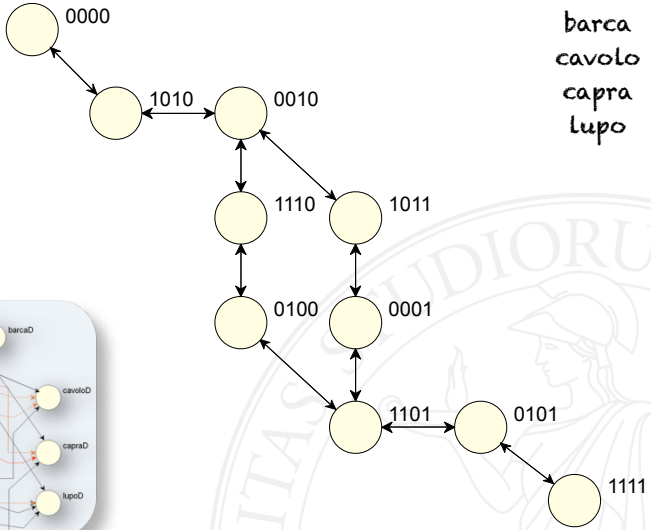


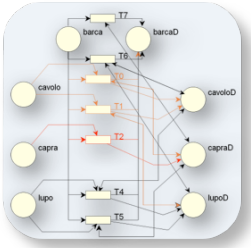
Il lupo... è un cavolo!

Soluzione: osservazioni

- Abbiamo dato regole per passare da una riva all'altra.
  - Il passaggio inverso ha regole speculari
- Risolve il problema?
  - Specifica tutti i movimenti possibili
  - Si puo' costruire grafo di raggiungibilita` e vedere se la marcatura richiesta e` raggiungibile e come.

Grafo di raggiungibilita`





Soluzione continua

- La rete andrebbe guidata…
- Si può osservare però che al fine di svolgere il compito alcune transizioni non hanno senso
  - barcaiolo solo da sinistra a destra…
  - bisognerebbe restringere anche possibilità passaggio capra da una riva all'altra (transizione T2)
- Si sono date le specifiche dei requisiti del problema, non la soluzione.
- Si e' detto **cosa** si puo' fare, non come farlo

Se usassi archi inibitori…

- Si riesce a risolvere il problema del guidare lo svolgimento della rete?

  - Falso problema… anche prima si riusciva ma non si era voluto
  - Non aumenta potenza in quanto la rete e` limitata

Estensione: Priorita`

- Alle transizioni è associata una priorità. Il prossimo scatto è scelto tra le transizioni abilitate a priorità più alta

- SVANTAGGIO: Si perde localita` di decisione della abilitazione di una transizione

Ripensare a soluzione barcaiolo

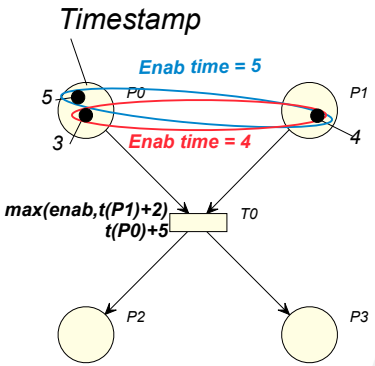
- Si riesce a risolvere il problema del guidare lo svolgimento della rete
  - Non aumenta potenza… pero`…
  - puo` essere un modo piu` elegante in quanto usiamo un attributo diverso per guida…
    - la topologia esprime solo i vincoli
    - le priorita` servono solo per disambiguare possibili conflitti

Time Basic nets (Ghezzi et al. 1989)

- Tempo associato alle transizioni

- Vengono associati:
  - degli insiemi di tempi di scatto possibili
  - definiti in maniera dinamica
  - come funzioni che possono fare riferimento a tempi assoluti e ai tempi dei singoli gettoni

TB net informalmente



I gettoni non sono anonimi (**timestamp**)

Tempo di abilitazione (enab) = il massimo tra i timestamp dei gettoni che compongono la tupla abilitante (**enabling tuple)**

Insieme dei tempi di scatto
Non possono essere minori del tempo di abilitazione (una transizione non può scattare prima di essere ablitata)

Tempo di scatto = scelto all'interno del set dei possibili… timestamp di tutti i gettoni prodotti

TB nets formalmente

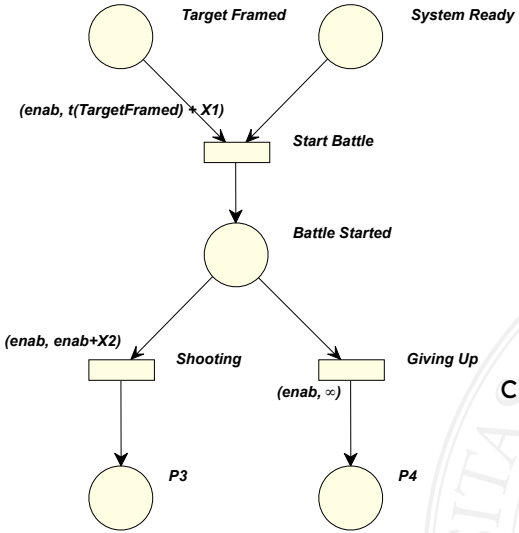
- Una rete TB è una 6-tupla $<P,T,\Theta;F,tf,m_0>$
- [P,T; F] come nelle reti di petri normali
- Θ è un insieme numerico (il dominio temporale)
- tf associa ad ogni transizione una funzione temporale tf_t.
  - dove $tf_t(en) \subseteq \Theta$,
    - dove en è una tupla abilitante
- m_0: $P \rightarrow \{ (\theta, mul(\theta)) \mid \theta \in \Theta \}$
  - è un multiset che esprime la marcatura iniziale

Semantica temporale debole (Weak) (informalmente)

- Una transizione può scattare solo in uno degli istanti identificati dalla sua funzione temporale
- Una transizione non può scattare prima di essere stata abilitata
- Una transizione anche se abilitata non è forzata a scattare

  - Utile per modellare eventi solo parzialmente definiti
    - Eventi che dipendono da componenti non modellati o modellizzabili
    - Una decisione umana, un guasto…

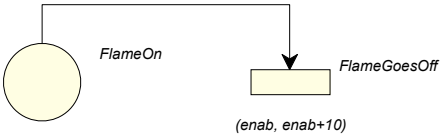
Esempio di WTS: figther



E` possibile sparare solo sotto certe condizioni…
L'esito della battaglia dipende da una decisione del pilota

Una sequenza ammissibile sotto WTS:
StartBattle(τ1), GivingUp (τ2) con τ2 - τ1 > X2
Cioè GivingUp scatta dopo il massimo dei tempi possibili di scatto di Shooting

Esempio di WTS: gas burner



Un guasto (la fiamma si spegne a causa del vento) è possibile ma potrebbe anche non accadere mai

Al contrario dei modelli stocastici non ci interessa modellare con quale probabilità potrà accadere… ma solo che può accadere

Axiom 1: Monotonicità rispetto alla marcatura iniziale

- Tutti i tempi di scatto di una sequenza di scatto devono essere non minori di uno qualunque dei timestamp dei gettoni della marcatura iniziale

- La marcatura deve essere consistente: cioè non deve contenere gettoni prodotti nel futuro

Axiom 2: Monotonicità dei tempi di scatto di una sequenza

- Tutti i tempi di scatto di una sequenza di scatti devono essere ordinati nella sequenza in maniera monotonicamente non decrescente
- Consistenza con la proprietà intuitiva:
  - il tempo non torna indietro…
- Due o più transizioni possono scattare nello stesso istante:
  - Azioni concorrenti simultanee
  - Granularità temporale più piccola delle necessità del modello

Axiom 3: Divergenza del tempo (non-zenonicità)

- Non è possibile avere un numero infinito di scatti in un intervallo di tempo finito

- Consistenza rispetto a proprietà intuitiva del tempo:
  - Il tempo avanza
    - non si può fermare
    - Non è suddivisibile in infinitesimi

WTS and MWTS

- Sequenze di scatti che soddisfano gli assiomi 1 e 3 sono chiamate sequenze ammissibili in semantica debole (WTS)

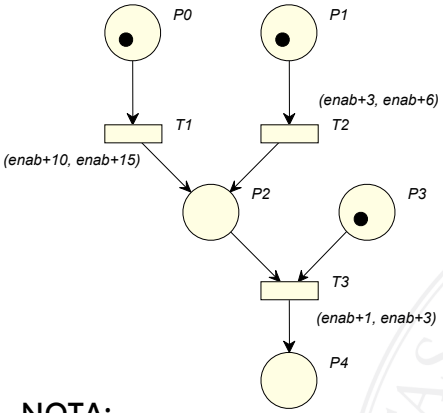
- Sequenze di scatti che soddisfano gli assiomi 1, 2 e 3 sono chiamate sequenze ammissibili in semantica monotonica debole (MWTS)

Theorem WTS ≡ MWTS

- Per ogni sequenza di scatti debole s esiste una sequenza di scatti monotonica debole ottenibile per semplice permutazione delle occorrenze degli scatti.

- Tecniche di analisi per (high-level) Petri net possono essere usate per reti con semantica debole

Esempio di equivalenza



Una possibile sequenza WTS:
(assumendo timestamp iniziali tutti uguali a zero)

- T1 scatta al tempo 12
- T3 scatta al tempo 14
- T2 scatta al tempo 4

La equivalente MWTS:

- T2 scatta al tempo 4
- T1 scatta al tempo 12
- T3 scatta al tempo 14

NOTA:
Nella marcatura prodotta dallo scatto di T2,
T3 risulta abilitata tra 5 e 7, ma non scatta!!

Semantica temporale forte (strong) (informalmente)

- Una transizione può scattare solo in uno degli istanti identificati dalla sua funzione temporale
- Una transizione non può scattare prima di essere stata abilitata
- Una transizione DEVE scattare ad un suo possibile tempo di scatto a meno che non venga disabilitata prima del proprio massimo tempo di scatto ammissibile

Semantica più diffusa (utile per i sistemi deterministici)
Default in molti modelli temporizzati (TPNs)

Assioma 4: Marcatura forte iniziale

- Il massimo tempo di scatto di tutti le abilitazioni nella marcatura iniziale deve essere maggiore o uguale del massimo timestamp associato ad un gettone della marcatura

- La marcatura iniziale deve essere consistente con la nuova semantica.
- Il gettone non avrebbe potuto essere creato a un istante successivo a quel timestamp senza che prima fosse scattata la transizione (entro il suo tempo massimo)

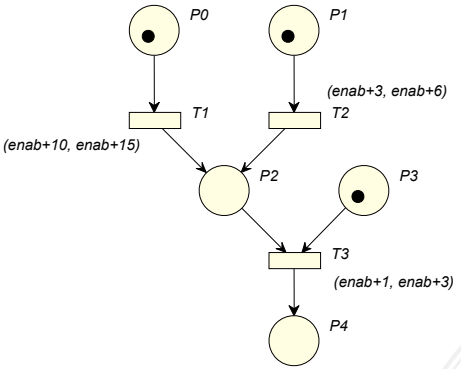
Assioma 5: Sequenza di scatti forte (STS)

- Una sequenza di scatti MWTS che parta da una marcatura forte iniziale è una sequenza di scatti forte se per ogni scatto il tempo di scatto della transizione non è maggiore del massimo tempo di scatto di una altra transizione abilitata.

- Una transizione abilitata DEVE scattare entro il suo massimo tempo di scatto, se non viene disabilita prima da un altro scatto

- Sequenze di scatto che soddisfano gli assiomi 1,2,3,4, e 5 sono dette sequenze ammissibili in semantica forte (STS)

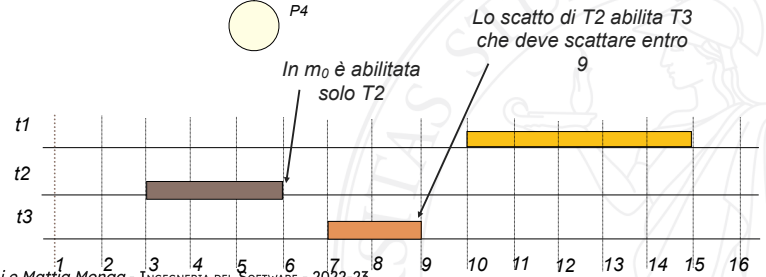
STS ≠ WTS



La sequenza MWTS:

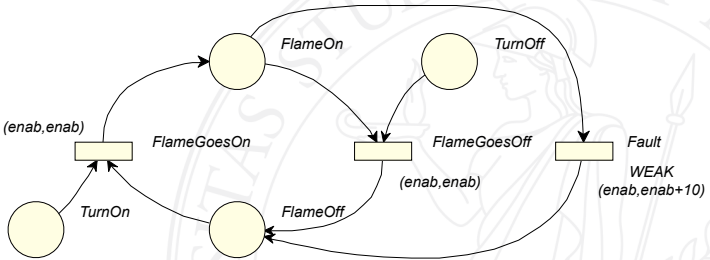
- t2 scatta al tempo 6
- t1 scatta al tempo 12
- t3 scatta al tempo 14

Non è una sequenza STS

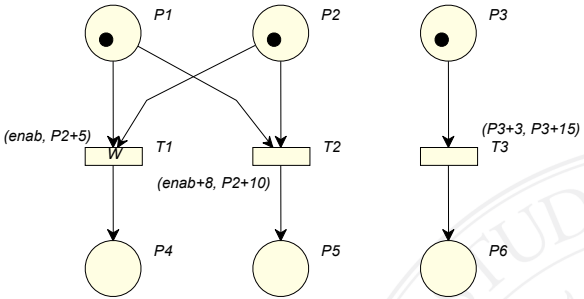


Mixed time semantics

- La semantica forte o debole viene associata alle singole transizioni invece che all'intera rete
  - Transizioni forti devono scattare entro il loro tempo massimo a meno che non vengano disabilitate prima
  - Transizioni deboli possono scattare entro il loro insieme di tempi di scatto

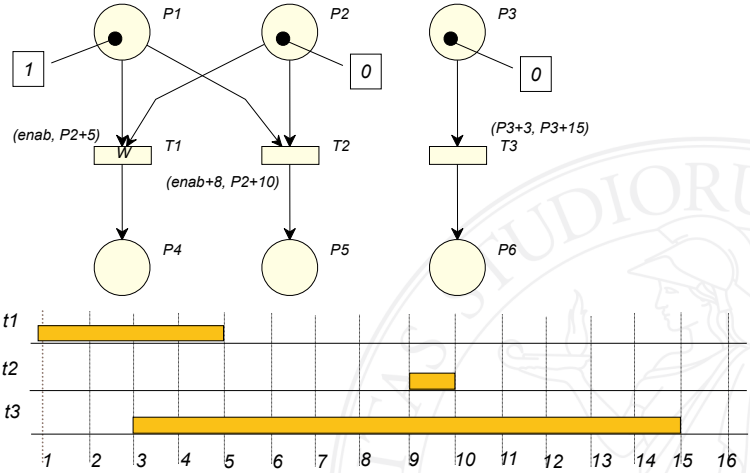


Un esempio

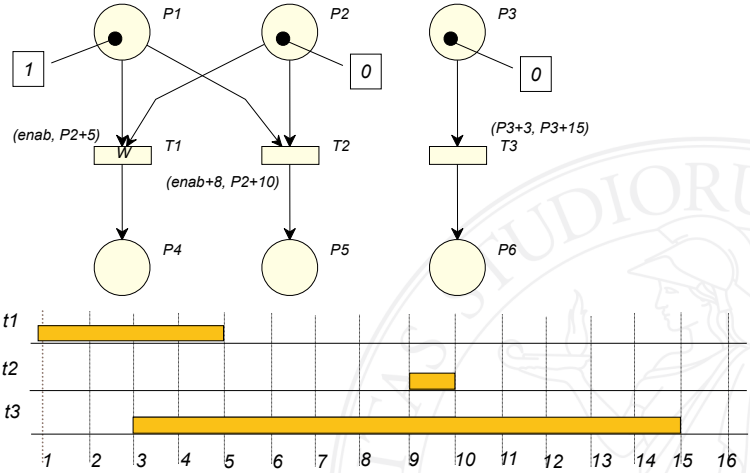


$$tf_{T1}(P1, P2) = \{\tau \mid max(P1, P2) \leq \tau \leq P2 + 5\}$$
$$tf_{T2}(P1, P2) = \{\tau \mid max(P1, P2) + 8 \leq \tau \leq P2 + 10\}$$
$$tf_{T3}(P3) = \{\tau \mid P3 + 3 \leq \tau \leq P3 + 15\} \quad .$$
$$m(P1) = \{1\};\ m(P2) = \{0\};\ m(P3) = \{0\}$$

WTS: analisi di abilitazione locale



Mixed TS: influenze globali



Modellare con reti TB

- Modellare un passaggio a livello con una rete di Petri