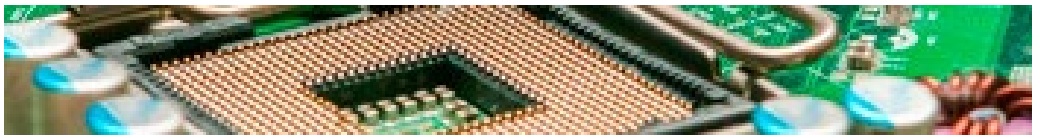
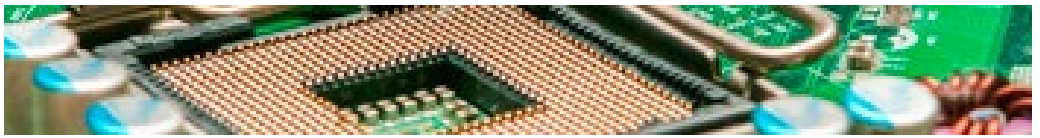
Tema 2: Planificación inteligente

2.2. Planificación jerárquica (HTN)



Contenido

1. Introducción
2. Ejemplo
3. Método
4. Tarea
5. Operador
6. Proceso de planificación
   1. Backtracking
   2. Ejemplo

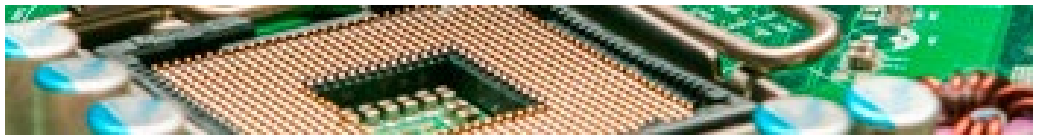


Introducción

Redes de tareas jerárquicas

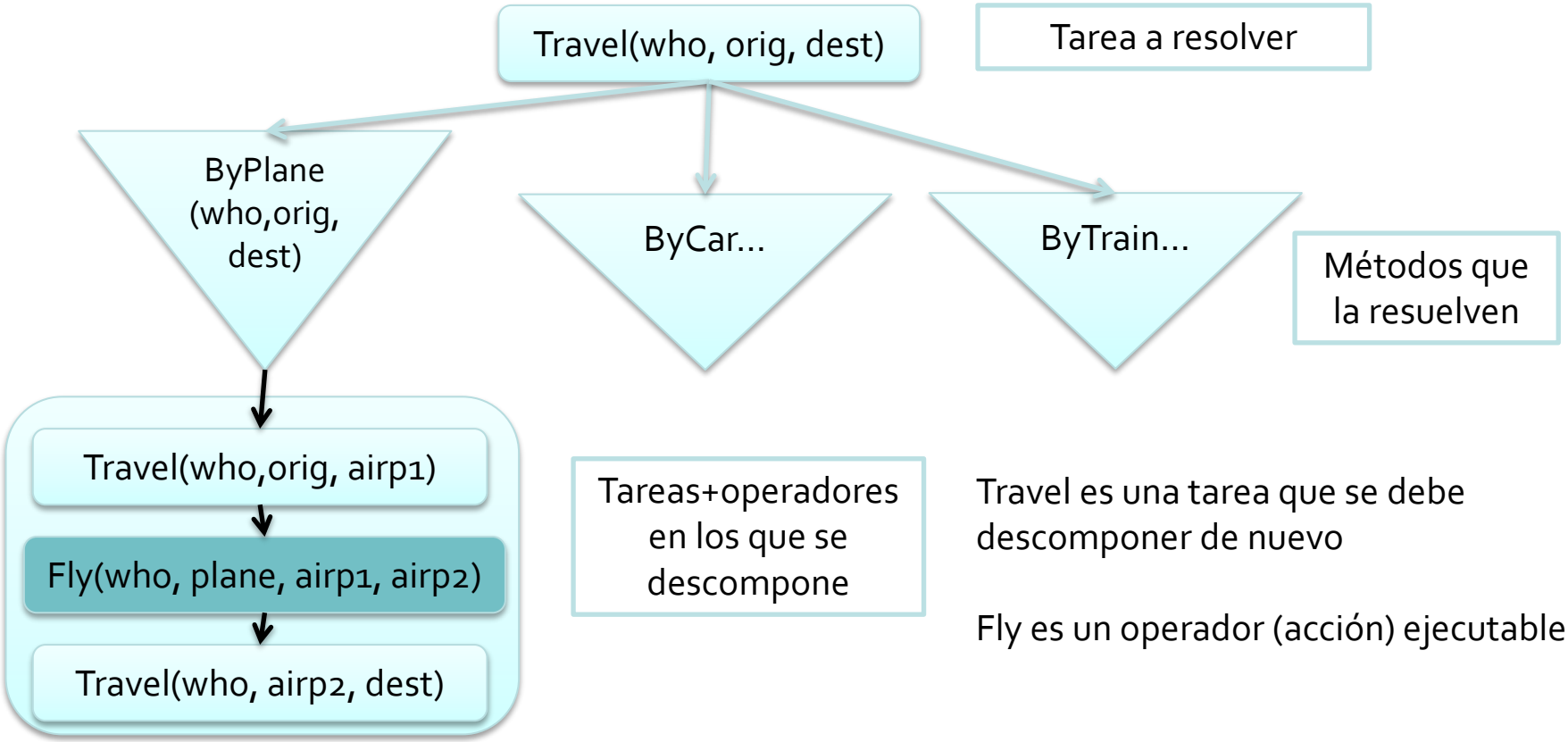
- Hierarchical Task Network (**HTN**) planning
- La idea fundamental es proceder de una forma más parecida a como los humanos resolvemos los problemas complejos
  - Primero los aspectos generales (tareas) y después los detalles

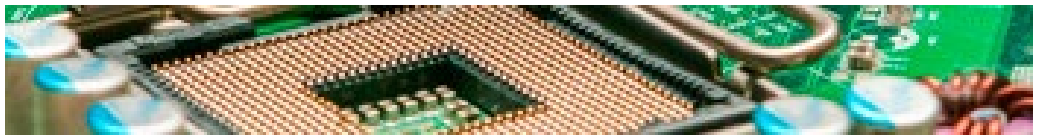
- Se definen **métodos abstractos** que se deben **descomponer** para resolver el problema
- Se parte desde un objetivo de alto nivel para construir una red de operadores primitivos, que representan acciones **ejecutables** del plan



Método abstracto y descomposición

Para viajar a un destino, se puede elegir entre viajar en avión, en coche o en tren:

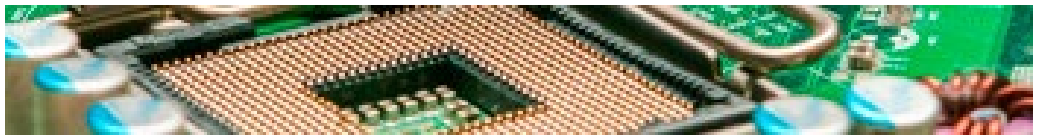




Método

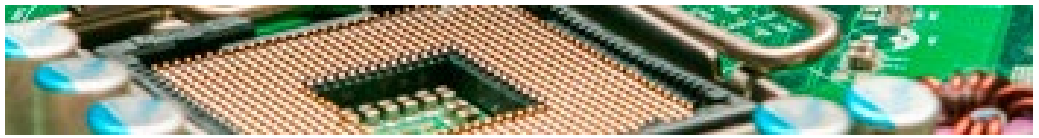
ByPlane (who,orig, dest)

- Representan una **forma distinta** de resolver una tarea
- No tienen por qué realizarse todos los métodos de los que está compuesta una tarea. Se escogerá el más apropiado en función del estado del mundo actual
- Están formados por un conjunto de tareas y operadores (acciones) que hay que ejecutar y/o alcanzar en un orden establecido para lograr resolver la tarea de nivel superior
- Estas tareas y operadores se organizan de forma secuencial



Componentes:

- Nombre: identificador del método
- Parámetros: lista de parámetros *heredados* del objetivo/tarea
- Precondiciones: lista opcional de condiciones sobre los parámetros que deben ser ciertas en el estado del mundo justo antes de la ejecución del método
- Red de tareas: conjunto de tareas u operadores asociados a un método - descomposición



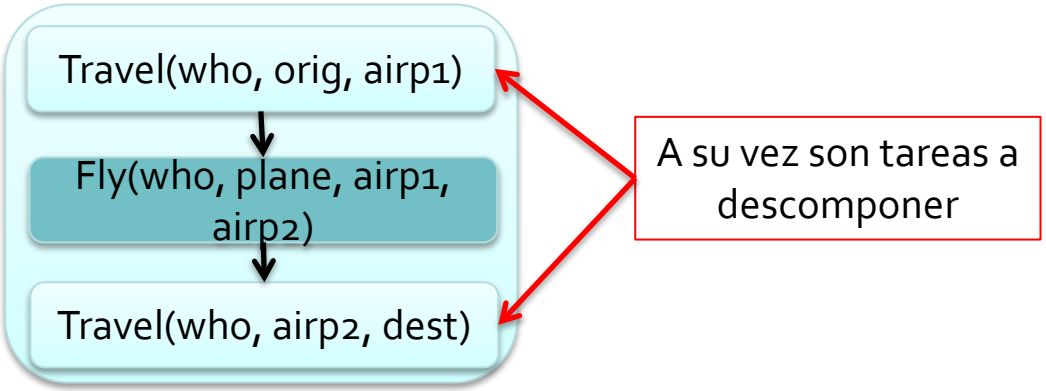
Ejemplo de método

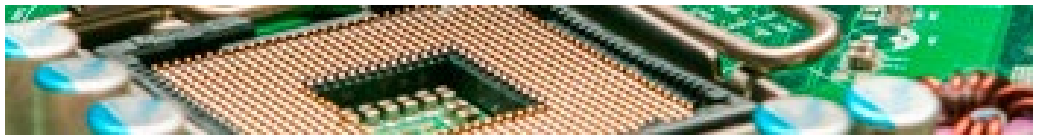
ByPlane:

Parámetros: *who: Person*
origin: Location
destination: Location

Condiciones: *airport[origin] != None and*
airport[destination] != None and
cash[who] >= plane_rate(origin, destination)

Red de tareas:



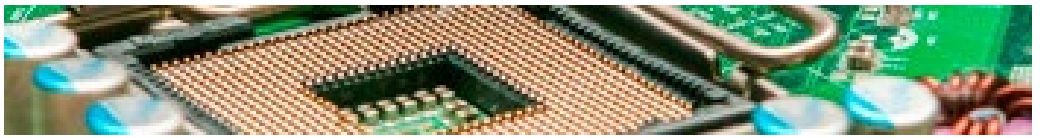


Operador

Representan las acciones que se pueden llevar a cabo dentro del mundo para poder alcanzar los distintos objetivos (**acciones ejecutables** que producen cambios en el estado del mundo)

Componentes:

- Nombre: identificador del operador
- Parámetros: utilizados en las condiciones y los efectos
- Condiciones: lista opcional de condiciones sobre los parámetros que deben ser ciertas en el estado del mundo justo antes de la realización de la tarea
- Efectos: especificación completa de los cambios que produce el operador sobre el estado del mundo



Ejemplo de operador

Fly:

Parámetros: *who: Person*
plane: Plane
from: Airport
to: Airport

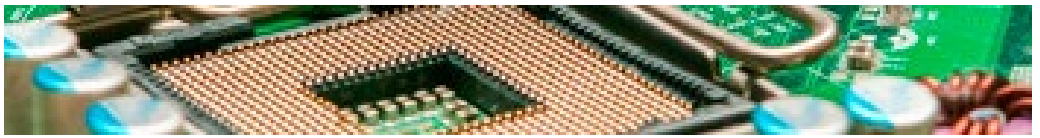
Fly(who, plane, airp1, airp2)

Condiciones:

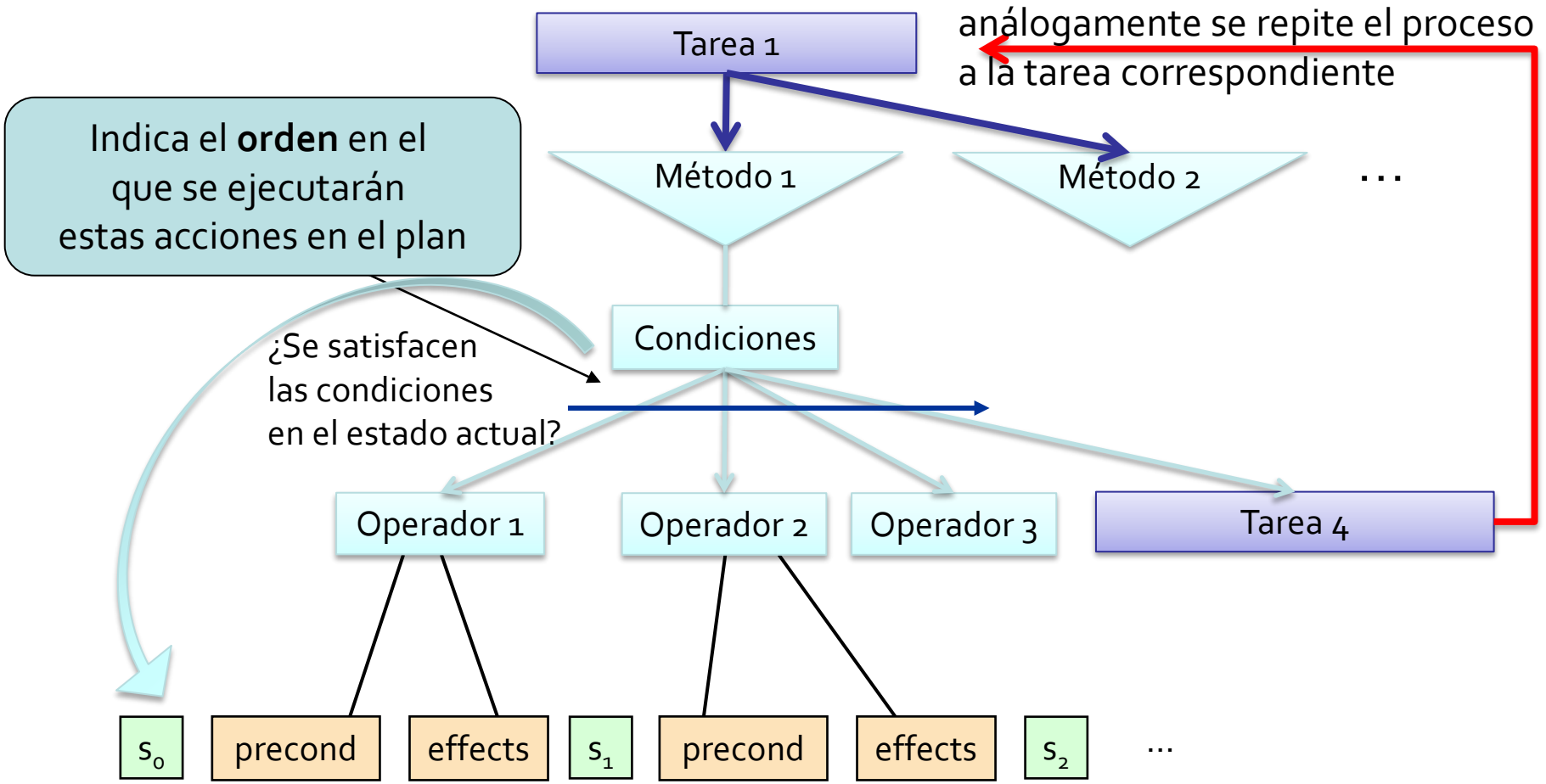
at[who] == at[plane] and
at[plane] == from

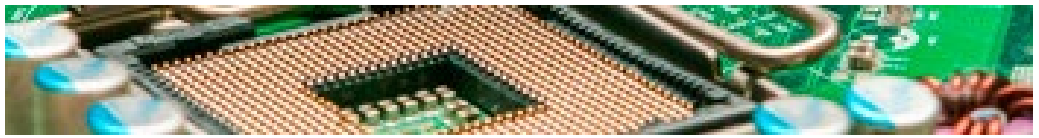
Efectos:

at[who]=to
at[plane]=to



Proceso de planificación

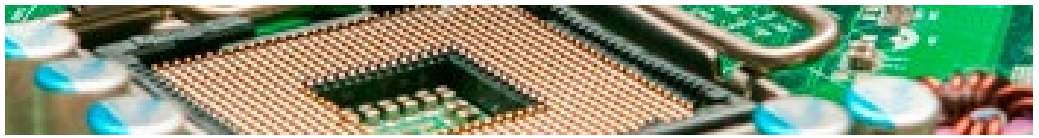




Proceso de planificación

- Se realiza un proceso de búsqueda entre todas las tareas, métodos y operadores definidos. Se basa en la descomposición de objetivos en tareas/operadores usando una búsqueda primero en profundidad.
- El proceso de búsqueda descompone de forma recursiva las tareas del objetivo que se está tratando, guiado por los métodos de descomposición hasta llegar a los niveles inferiores.

- Cuando se alcanza un operador, se comprueban sus precondiciones en el estado del mundo actual y, si son ciertas, el operador se añade al proceso y el estado actual se actualiza en función de lo expresado en los efectos del operador.
- Si el operador no se puede ejecutar en el estado del mundo actual, la descomposición por el método falla y el planificador debe probar con otro método: uso de técnica de **backtracking** – estrategia de búsqueda.

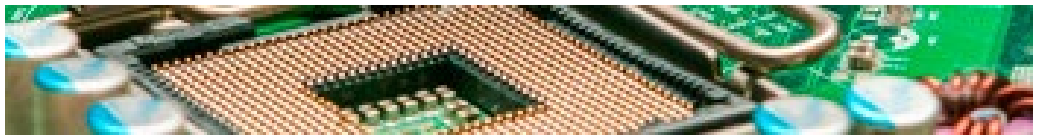


Backtracking

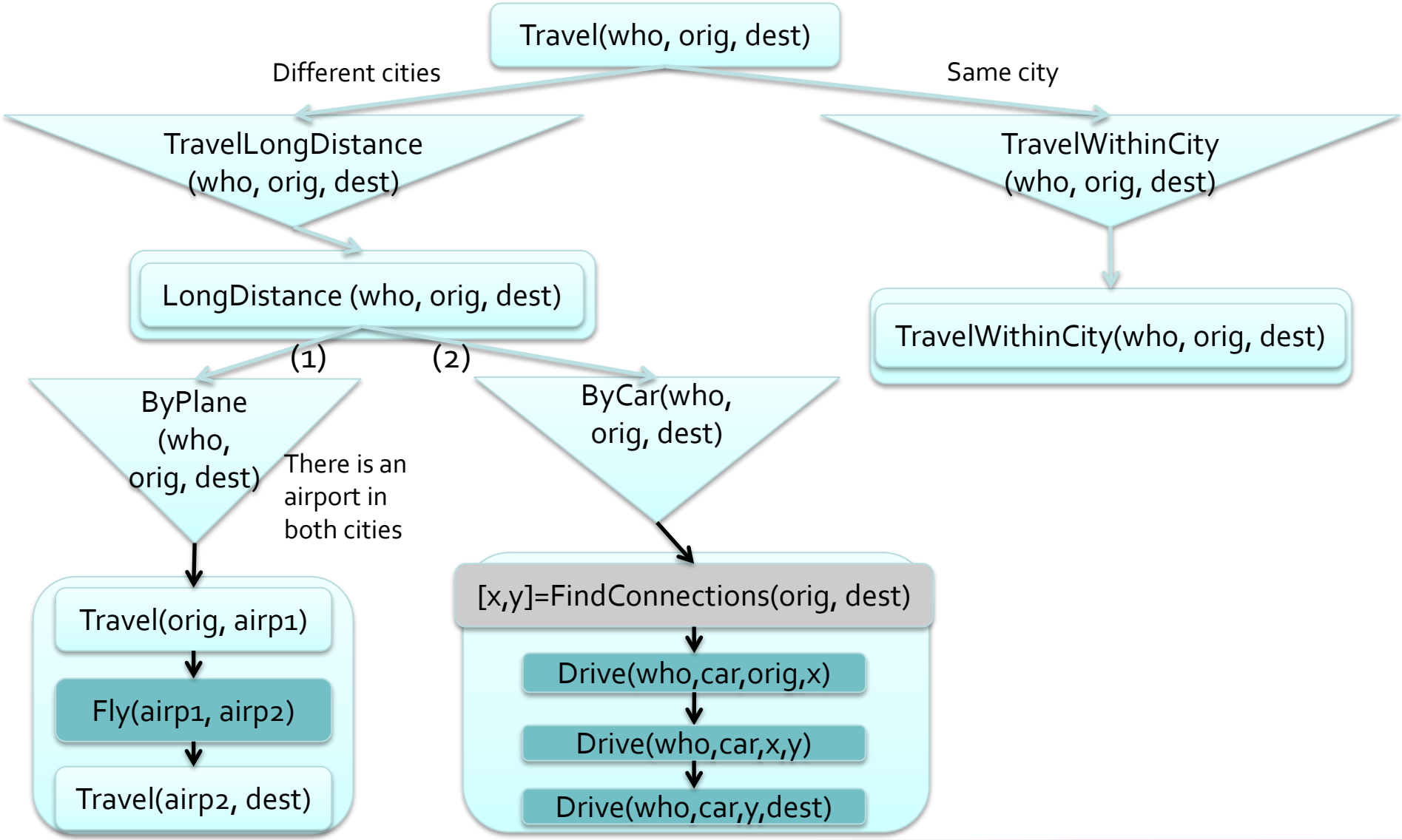
Proceso mediante el cual se vuelve atrás en el árbol de decisión al elemento justo anterior y al instante de tiempo en el que fue evaluado. Se deshacen todos los cambios que se dieron en el estado del mundo hasta volver justo al escenario del elemento anterior.

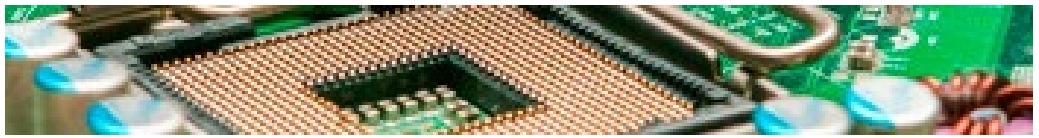
Se produce cuando:

- No se puede realizar ninguna unificación de los objetos disponibles en el estado del mundo.
- No se puede cumplir con las condiciones definidas sobre un parámetro concreto.

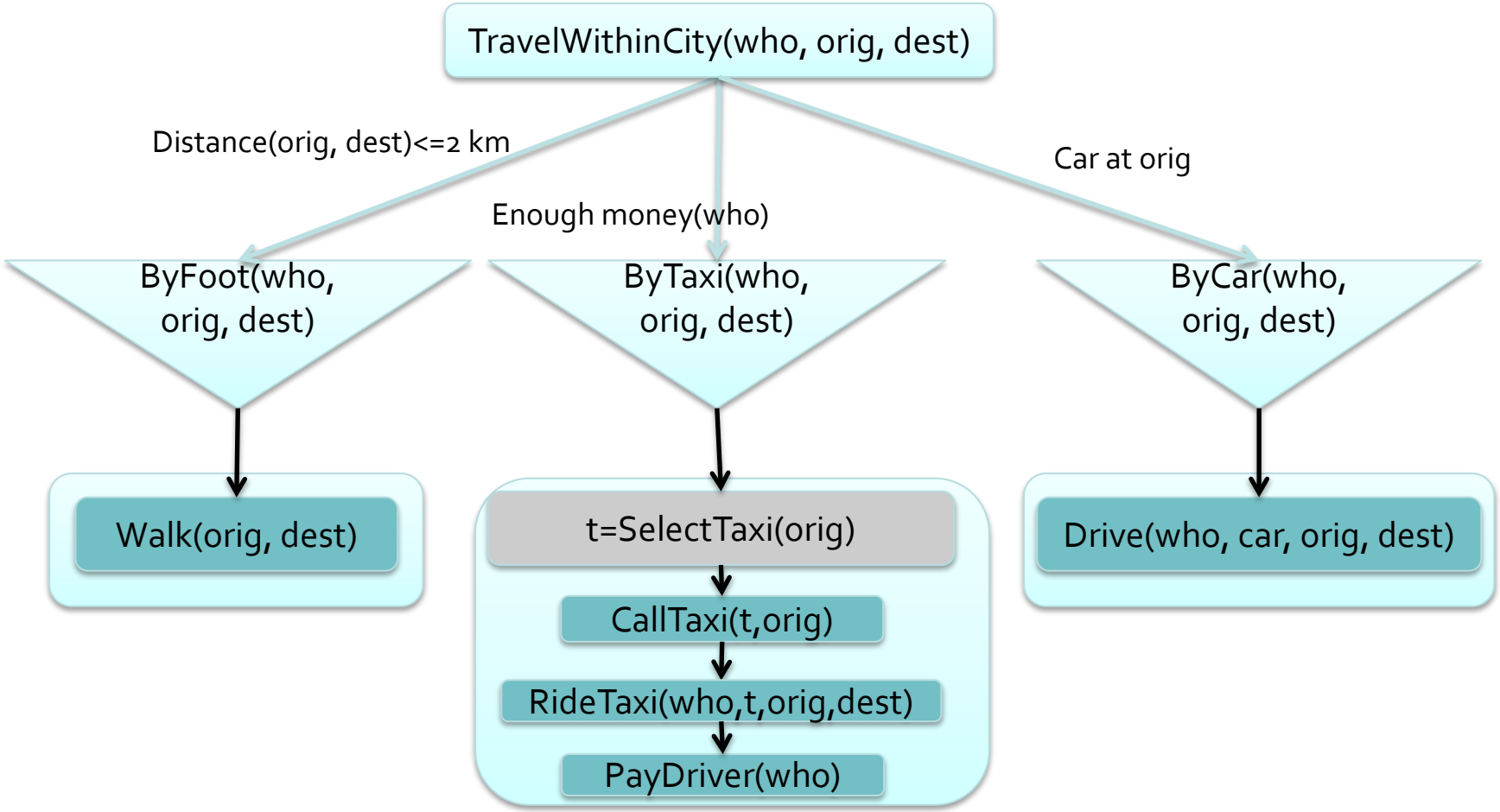


Ejemplo completo (1)





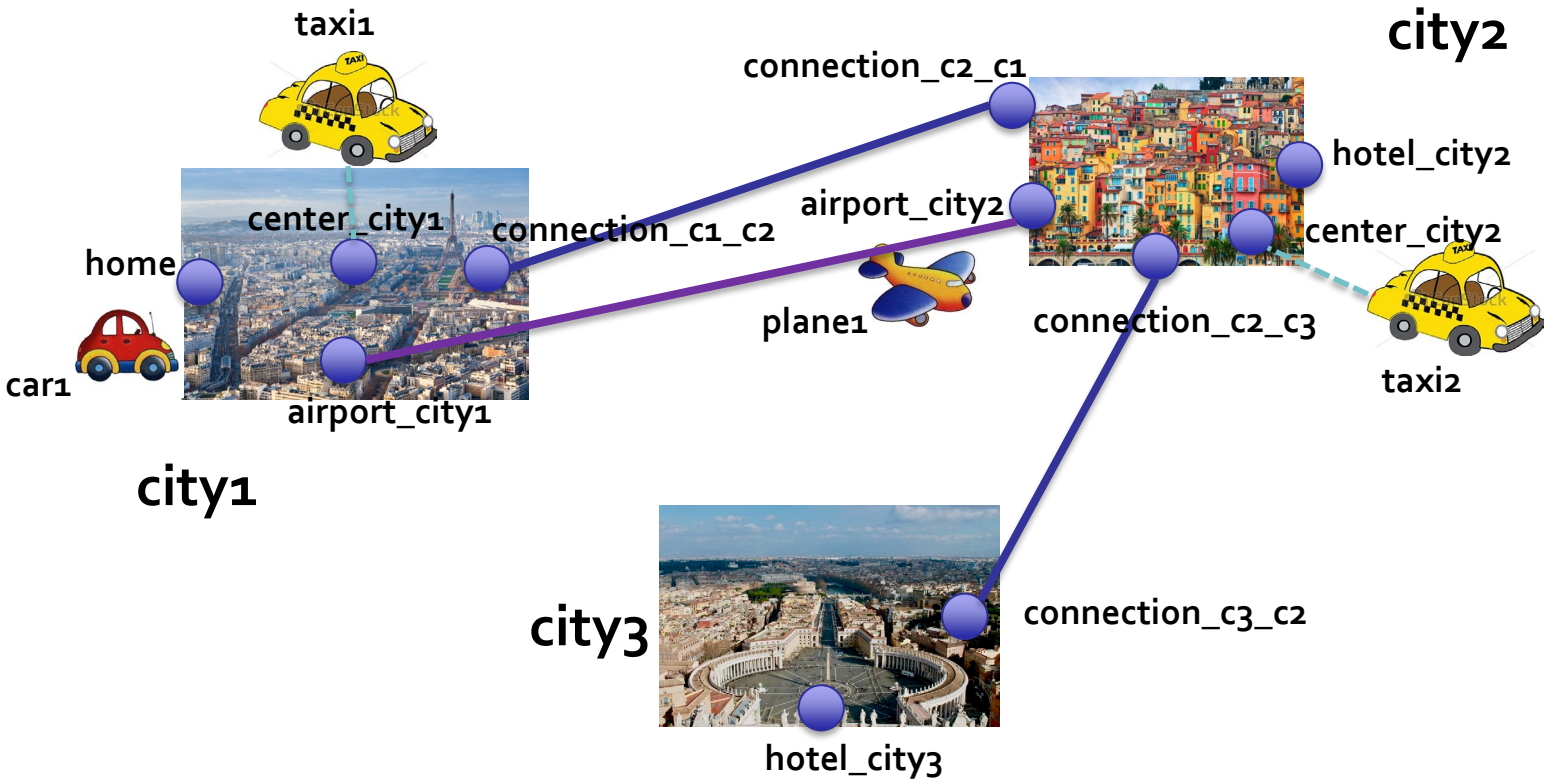
Ejemplo completo (2)



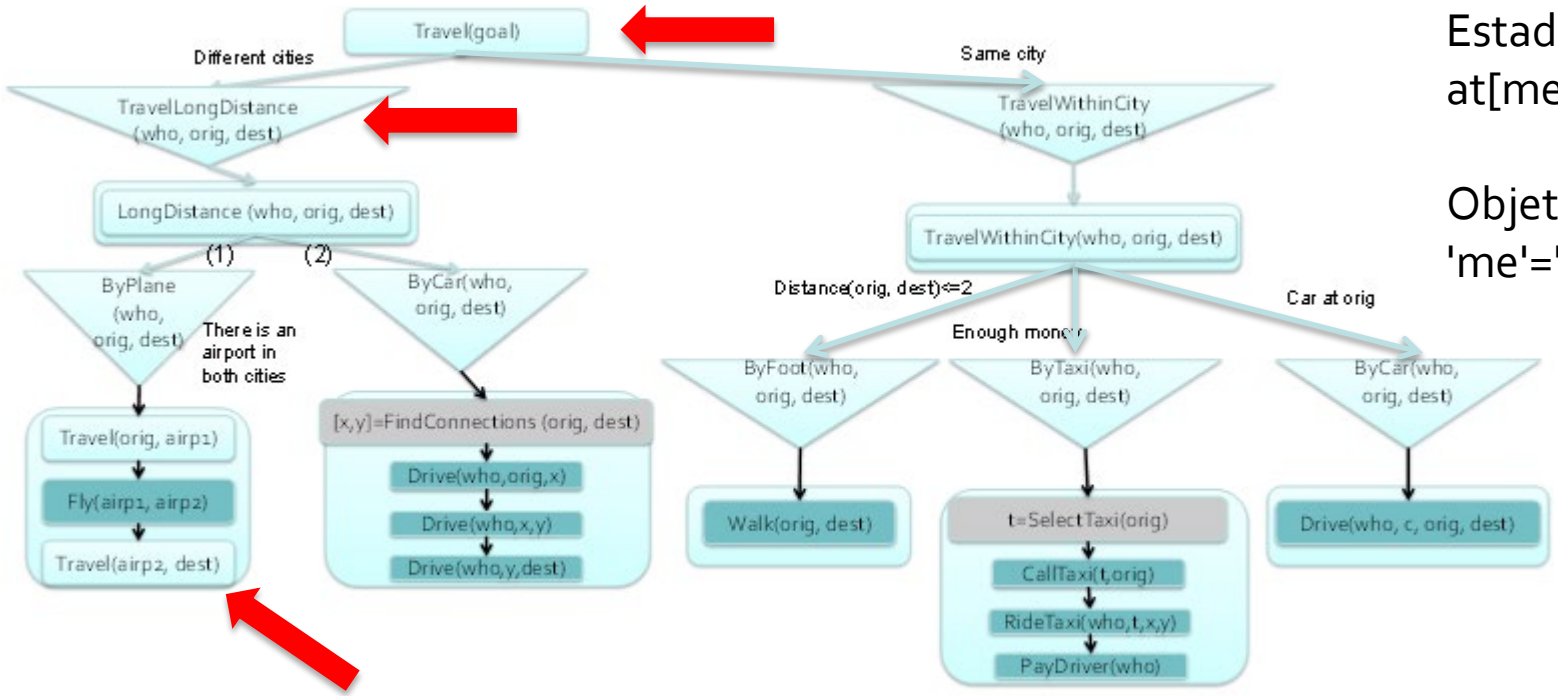
Ejemplo del proceso de planificación

Escenario con tres ciudades conectadas entre sí por diferentes medios y con diferentes localizaciones

- Las localizaciones dentro de una ciudad están completamente conectadas entre sí
- En la figura solo se muestran las conexiones (*connection*) entre ciudades



Ejemplo del proceso de planificación

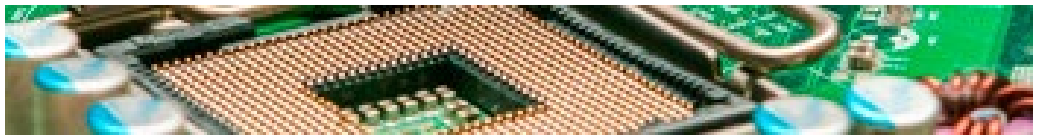


Estado inicial:
at[me]=home

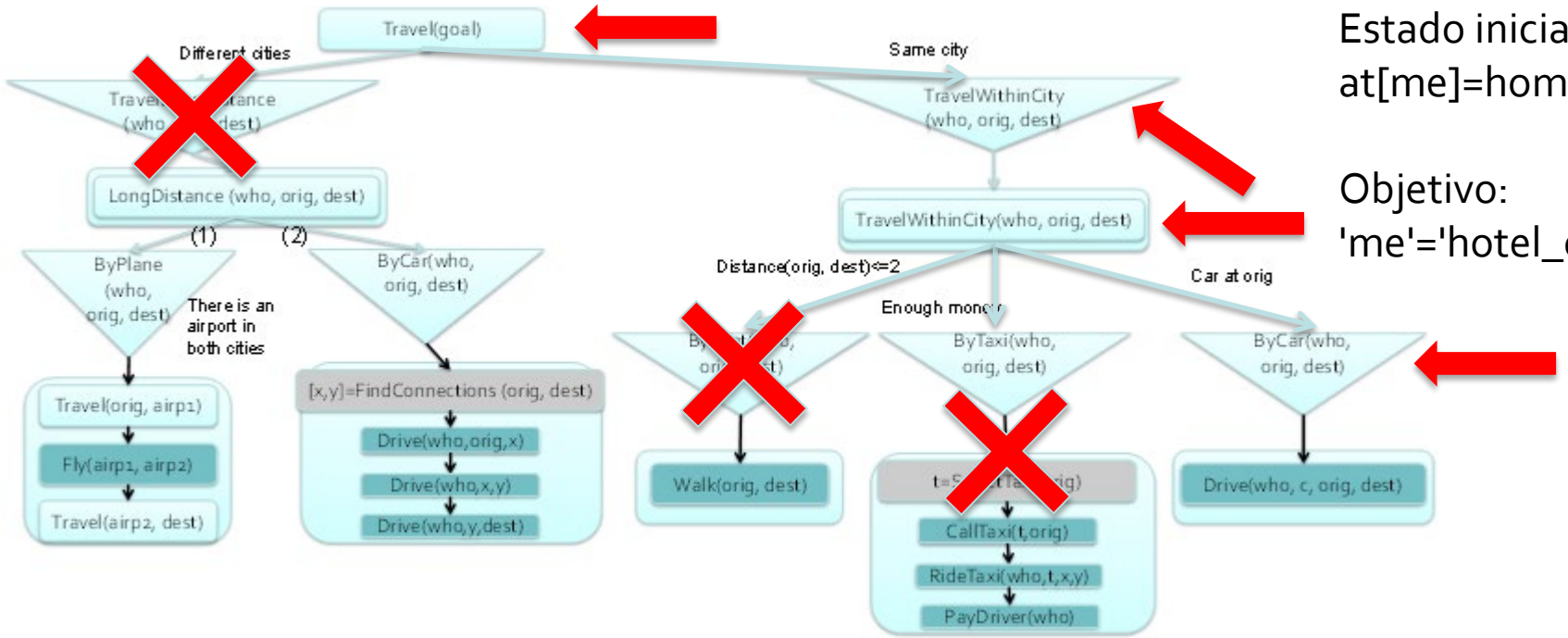
Objetivo:
'me'='hotel_city2'

1. depth 0 tasks [('travel', {**'me'**:**'hotel_city2'**})] Hay que descomponerlo...
2. depth 1 tasks [('travel_long_distance', 'me', 'home', 'hotel_city2')]
3. depth 2 tasks [('travel', {'me':'airport_city1'}), ('fly', 'me', 'plane1', 'airport_city1', 'airport_city2'), ('travel', {'me':'hotel_city2'})]

Hay que descomponerlo...



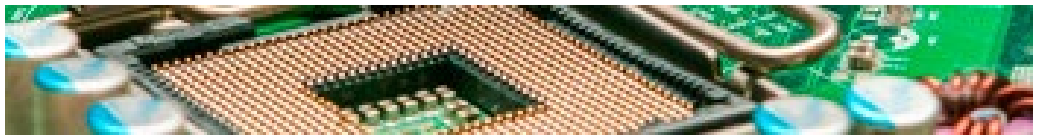
Ejemplo del proceso de planificación



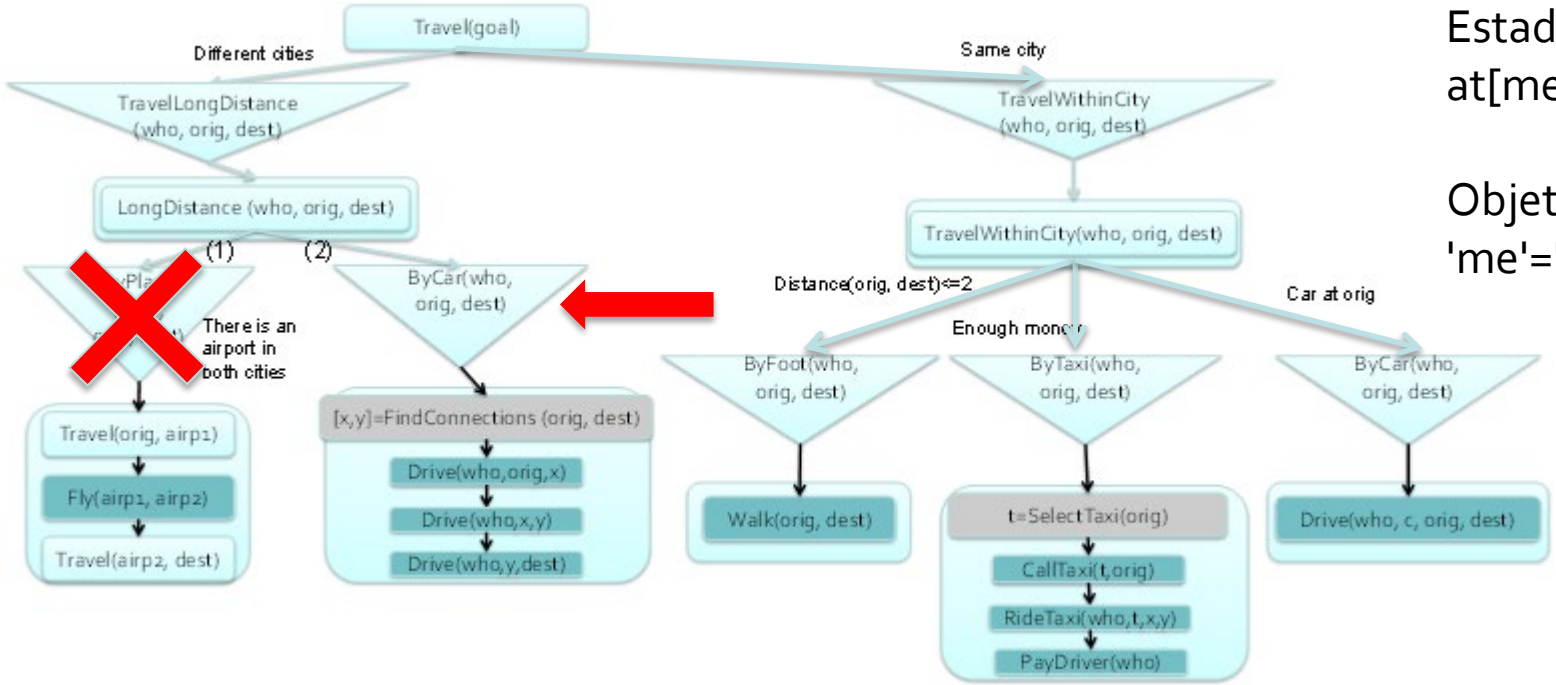
Estado inicial:
at[me]=home

Objetivo:
'me'='hotel_city2'

1. depth 2: Resolviendo ('travel', {'me':'airport_city1'}) → TravelLongDistance falla porque *home* está en la misma ciudad que el *airport_city1*. Por tanto, utilizamos TravelWithinCity
2. depth 3 tasks [('travel_within_city', 'me', 'home', 'airport_city1'), ('fly', 'me', 'plane1', 'airport_city1', 'airport_city2'), ('travel', {'me':'hotel_city2 '})]
3. depth 3: Resolviendo ('travel_within_city', 'me', 'home', 'airport_city1') → ByFoot falla porque la distancia es mayor que 2. ByTaxi falla porque no hay taxi disponible que seleccionar en home → utilizamos ByCar
4. depth 4 tasks [('drive', 'me', 'car1', 'home', 'airport_city1'), ('fly', 'me', 'plane1', 'airport_city1', 'airport_city2'), ('travel', {'me':'hotel_city2 '})]



Ejemplo del proceso de planificación

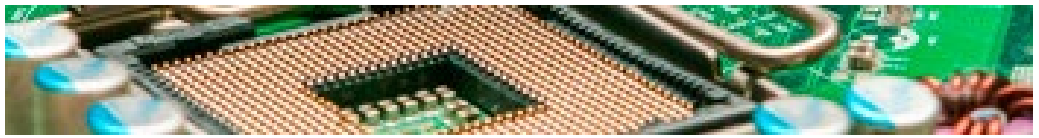


Estado inicial:
at[me]=home

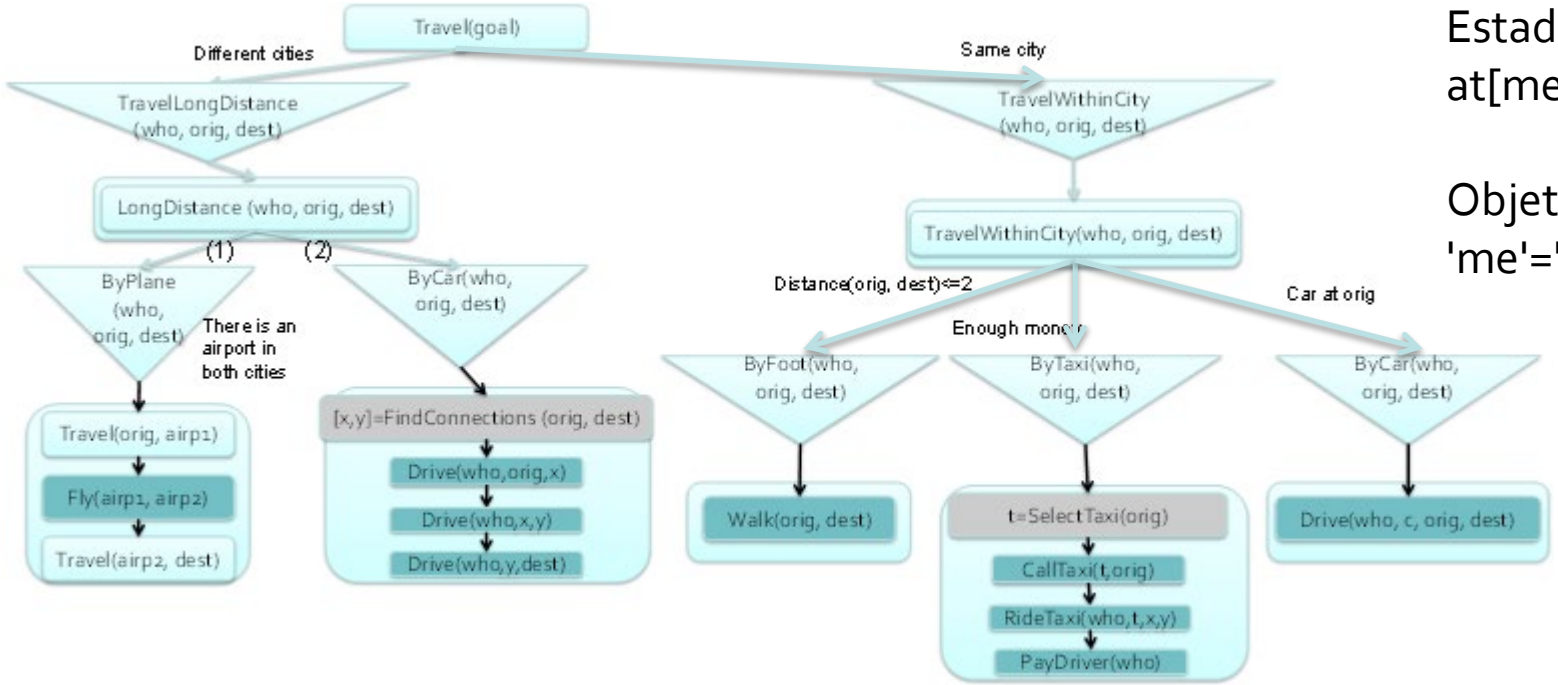
Objetivo:
'me'='hotel_city2'

depth 4 tasks [('drive', 'me', 'car1', 'home', 'airport_city1'), ('fly', 'me', 'plane1', 'airport_city1', 'airport_city2'), ('travel', {'me':'hotel_city2 '})]

1. depth 4 action ('drive', 'me', 'car1', 'home', 'airport_city1') → Al ejecutar el operador se genera un nuevo estado
   - state1.at = {**'me': 'airport_city1', 'car1': 'airport_city1',** 'plane1': 'airport_city2', 'taxi1': 'center_city1', 'taxi2': 'center_city2'}
2. depth 5 tasks [('fly', 'me', 'plane1', 'airport_city1', 'airport_city2'), ('travel', {'me':'hotel_city2 '})]
3. depth 5 action ('fly', 'me', 'plane1', 'airport_city1', 'airport_city2') → La acción no se puede aplicar porque no tenemos un avión en airport_city1 (y no hay más formas de "volar") → BACKTRACKING



Ejemplo del proceso de planificación

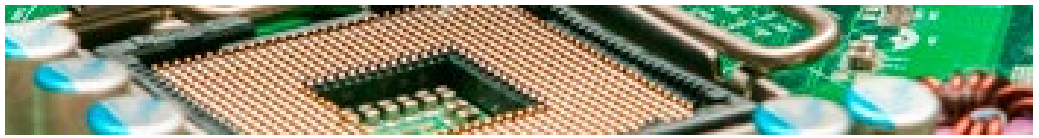


Estado inicial:
at[me]=home

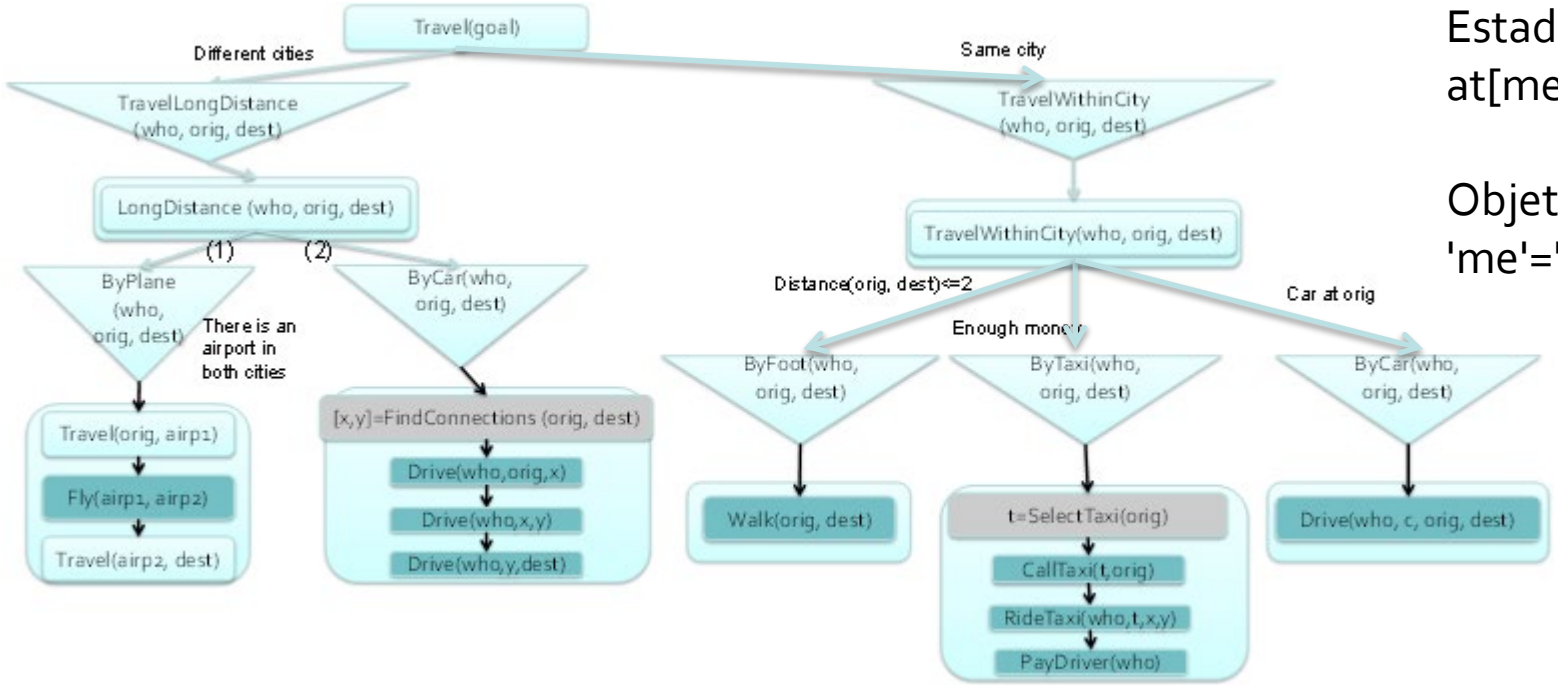
Objetivo:
'me'='hotel_city2'

Deshacemos la llamada recursiva (vía **backtracking**) y **regresamos** hasta depth 1 tasks.

Ahora probaremos la alternativa 2: ByCar('me', 'home', 'hotel_city2'), buscando un par de conexiones: FindConnections('home', 'hotel_city2')



Ejemplo del proceso de planificación

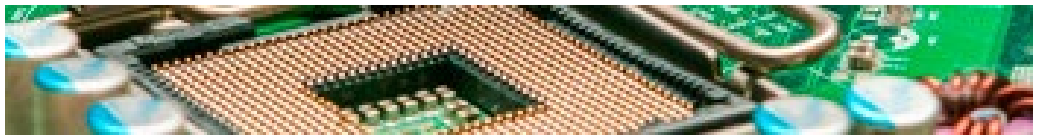


Estado inicial:
at[me]=home

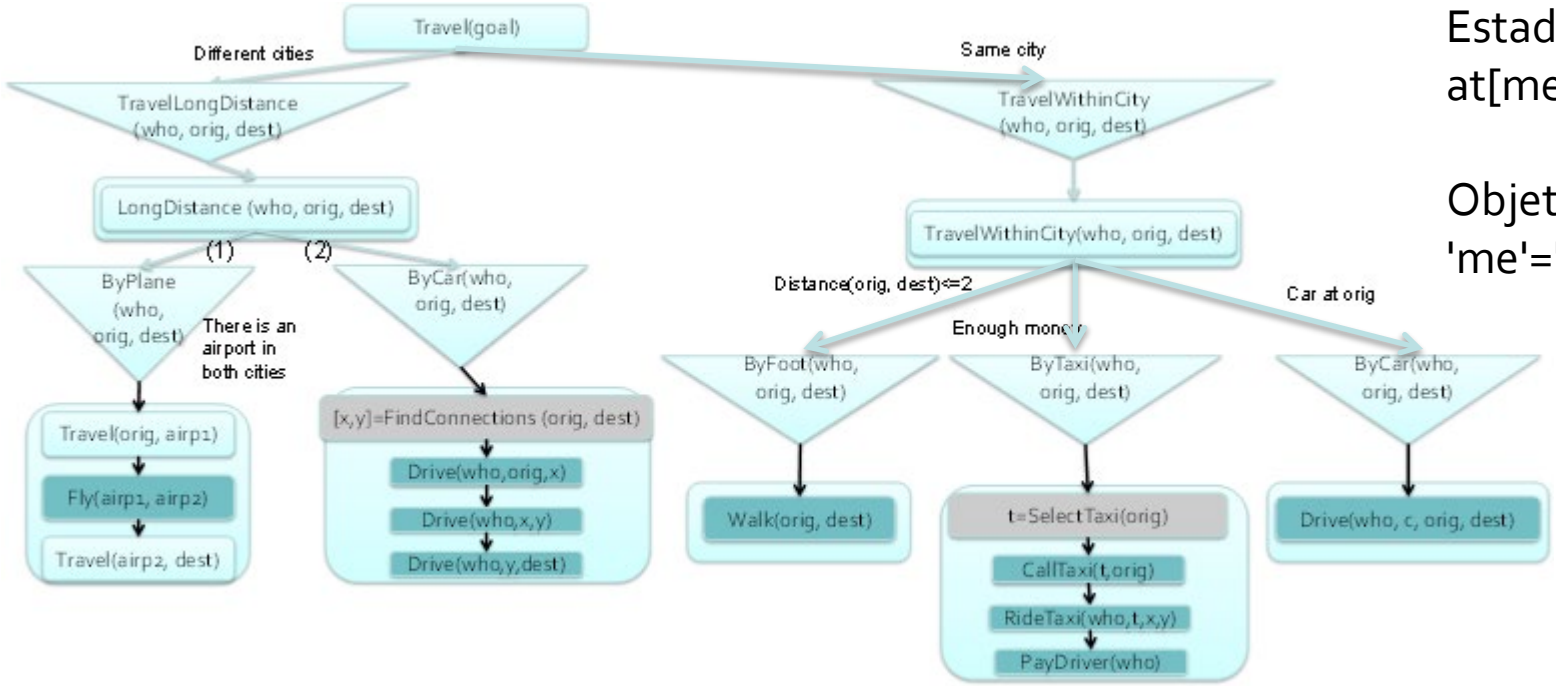
Objetivo:
'me'='hotel_city2'

Ahora probaremos la alternativa 2: ByCar('me', 'home', 'hotel_city2') con FindConnections('home', 'hotel_city2')

1. depth 2 action ('drive', 'me', 'car1', 'home', 'connection_c1_c2')
   - state1.at = {'me': 'connection_c1_c2', 'car1': 'connection_c1_c2', 'plane1': 'airport_city2', 'taxi1': 'center_city1', 'taxi2': 'center_city2'}
2. depth 3 action ('drive', 'me', 'car1', 'connection_c1_c2', 'connection_c2_c1')
   - state1.at = {'me': 'connection_c2_c1', 'car1': 'connection_c2_c1', 'plane1': 'airport_city2', 'taxi1': 'center_city1', 'taxi2': 'center_city2'}



Ejemplo del proceso de planificación



Estado inicial:
at[me]=home

Objetivo:
'me'='hotel_city2'

1. depth 4 action ('drive', 'me', 'car1', 'connection_c2_c1', 'hotel_city2')
   - state1.at = {'me': 'hotel_city2', 'car1': 'hotel_city2', 'plane1': 'airport_city2', 'taxi1': 'center_city1', 'taxi2': 'center_city2'}
2. depth 5 tasks []
   - depth 5 returns plan [('drive', 'me', 'car1', 'home', 'connection_c1_c2'), ('drive', 'me', 'car1', 'connection_c1_c2', 'connection_c2_c1'), ('drive', 'me', 'car1', 'connection_c2_c1', 'hotel_city2')]