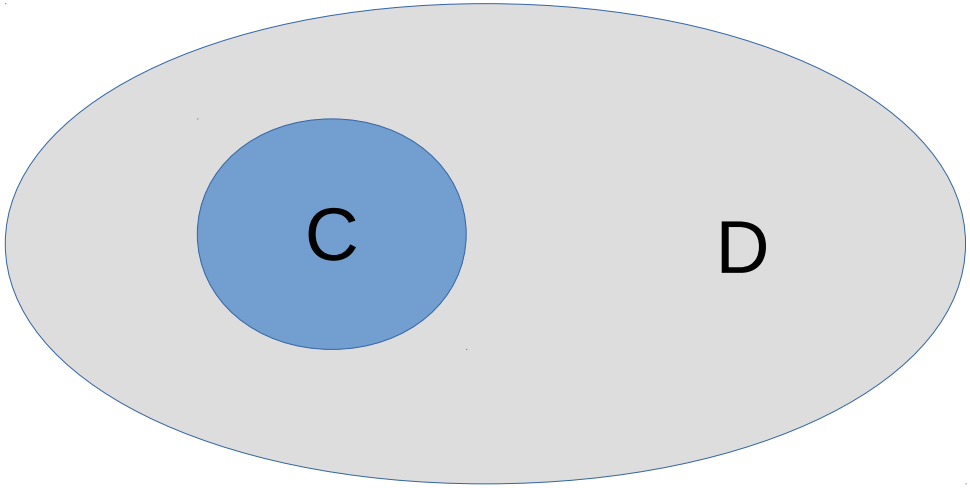
RDF: semantica

RDF ha una semantica basata su if

Se C è sottoclasse di D, allora le istanze di C devono essere un sottoinsieme delle istanze di D



Member(I,C) => Member(I,D)

Strumenti

- **Apache Jena**: framework Java per lo sviluppo di applicazioni orientate al web semantico. Permette di leggere e scrivere RDF
- **AllegroGraph**: triplestore (DB a triple) per RDF
- **Turtle, N3**: linguaggi di rappresentazione alternativi di RDF
- **SPARQL**: linguaggio di interrogazione per rappresentazioni RDF
- …

Applicazioni

- **Linked (Open) Data**
- **DBpedia**: wikipedia + annotazioni semantiche
- **CreativeCommons**: fornisce descrizioni delle licenze in RDF per consentire l'applicazione di tecniche di ragionamento automatico
- **FOAF**: friend of a friend ontology
- **Press Association**: agenzia stampa che pubblica notizie annotate semanticamente
- *Ecc.*

Esempio: CC

creative commons

Describing Copyright in RDF

Creative Commons Rights Expression Language

The Creative Commons Rights Expression Language (CC REL) lets you describe copyright licenses in RDF. For more information on describing licenses in RDF and attach descriptions to digital works, see CC REL in the Creative Commons wiki.

Classes

Work

a potentially copyrightable work

License

a set of requests/permissions to users of a Work, e.g. a copyright license, the public domain, information for distributors

Jurisdiction

the legal jurisdiction of a license

Permission

an action that may or may not be allowed or desired

Requirement

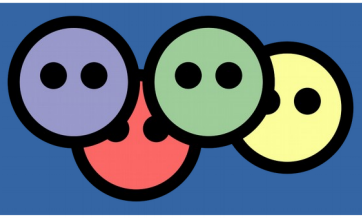
an action that may or may not be requested of you

Prohibition

something you may be asked not to do

Permissions

Esempio: FOAF



- Progetto finalizzato a linkare persone che usano il web Definisce un ricco vocabolario, il cui nucleo contiene le categorie:

Agent, Person
Name, title
Img, depiction (depicts)
FamilyName, givenName
Knows, based_near
Age, made (maker)
primaryTopic (primaryTopicOf)
Project, Organization
Group, member
Document
Image

Class: foaf:Agent

Agent - An agent (eg. person, group, software or physical artifact).

Status: stable

Properties include: gender yahooChatID account birthday icqChatID aimChatID jabberID made mbox interest tipjar skypeID topic_interest age mbox_sha1sum status msnChatID openid holdsAccount weblog

Used with: maker member

Has Subclass Group Person Organization

The Agent class is the class of agents; things that do stuff. A well known sub-class is Person, representing people. Other kinds of agents include Organization and Group.

The Agent class is useful in a few places in FOAF where Person would have been overly specific. For example, the IM chat ID properties such as jabberID are typically associated with people, but sometimes belong to software bots.

Fonte: http://xmlns.com/foaf/spec/#sec-glance

OWL 2: Web Ontology Language

- **OWL2:** linguaggio **dichiarativo** del semantic web ideato per **definire ontologie** tramite specifica di classi (categorie), proprietà, individui e valori
- Le ontologie OWL possono essere **pubblicate sul web** e **riferite da altre ontologie**, per costruire KB più complesse e raffinate

OWL 2: Semantica

- **Semantica definita formalmente**, permette di scrivere KB sulle quali si possono applicare inferenze in modo automatico
- OWL2 ha due semantiche:
  - **Semantica diretta**, si basa su logiche descrittive:
    - È applicabile a un frammento del linguaggio detto **OWL2 DL**
  - **Semantica che si basa su grafi RDF**:
    - È applicabile al linguaggio OWL intero, spesso indicato dal termine **OWL2 Full**

OWL 2: Semantica diretta

- È applicabile a un frammento del linguaggio detto **OWL2 DL**
- è compatibile con la **logica descrittiva SROIQ**. Le **logiche descrittive** sono logiche del prim' ordine specificamente customizzate per scrivere ontologie
- **Svantaggio**: restringe sintatticamente il linguaggio
- è descritta nel documento https://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/ (non è da studiare ma date un' occhiata, è breve).
- **Vantaggio**: è **decidibile**, cioè qualunque query trova sempre una risposta vero o falso

Approfondimento (opzionale)

- La **logica descrittiva SROIQ** è introdotta in questa presentazione:
- http://www.semantic-web-book.org/w/images/2/2d/W2011-06-DLs.pdf

- **S** linguaggio attributivo con negazione atomica, intersezione di ruoli, restrizioni universali, quantificazione esistenziale limitata, ruoli transitivi
- **R** role inclusion axioms con ruoli a complessità limitata, (ir)riflessività, disgiunzione di ruoli
- **O** classi enumerative con restrizioni sui valori degli oggetti
- **I** proprietà inverse
- **Q** restrizioni di cardinalità

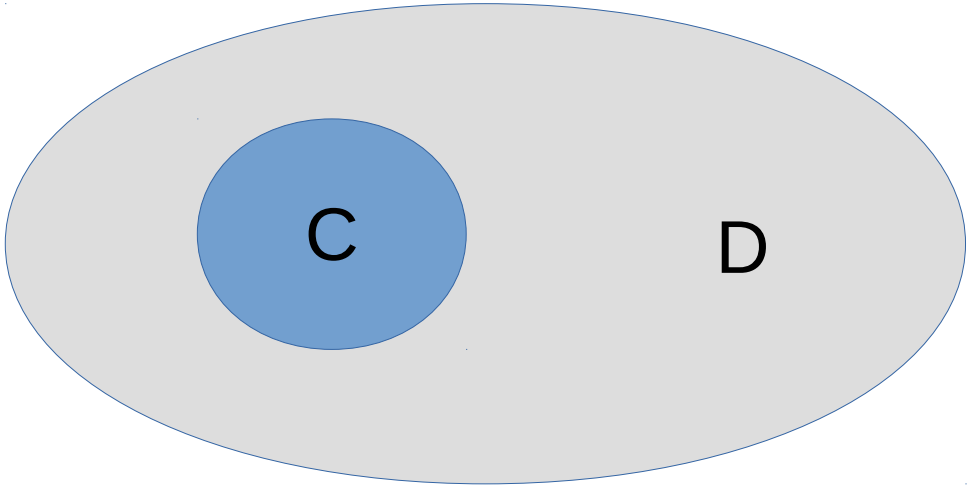
OWL 2: semantica basata su RDF

- **Estende la semantica formale di RDF con tutti gli elementi che permettono di gestire i costrutti introdotti da OWL 2**
- **Vantaggio**: consente di rappresentare conoscenza che è difficile espremere in OWL2 DL
- **Svantaggio**: si perde la decidibilità
- In alcuni domini applicativi sono stati definiti sottolinguaggi di OWL che coniugano i giusti elementi espressivi con dimostrazioni di soundness e completeness della logica necessaria a dare semantica al frammento di linguaggio identificato

OWL: semantica

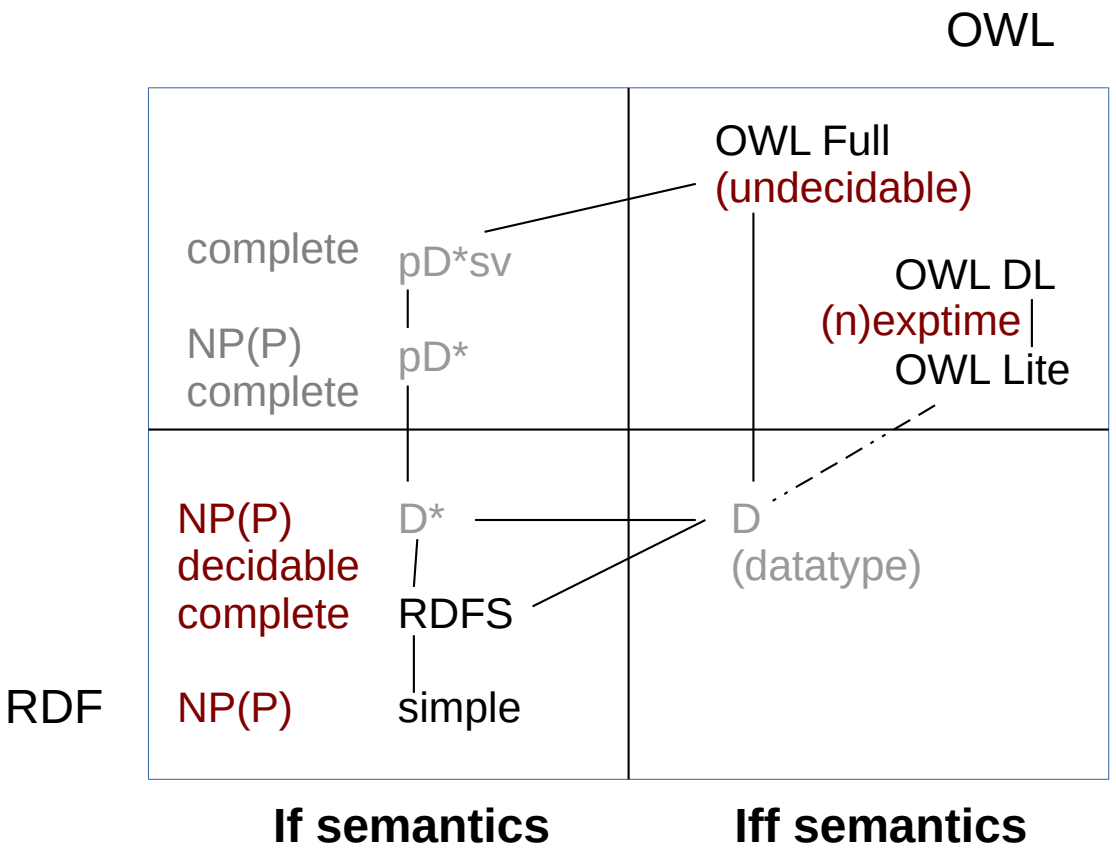
OWL ha una semantica basata su iff

C è sottoclasse di D **se e solo se** le istanze di C sono un sottoinsieme delle istanze di D



Member(I,C) ⇒ Member(I,D)
∃ I, Member(I, D) ⇒ Member(I, C)

Linguaggi e decidibilità



Inferenza in linguaggi RDF e OWL

RDF ha una semantica basata su if

OWL su if and only if
Permette di fare più inferenze.
È più forte

Le linee connettono linguaggi con inferenze che estendono quelle di altri: il linguaggio sopra estende l'inferenza di quello sotto, quindi permette di trarre tutte le conclusioni che sotto sono derivabili più altre originariamente non derivabili
Es. RDFS estende RDF, OWL DL estende OWL Lite, ...

Da: H. J. ter Horst, Web Semantics: Science, Services and Agents on the World Wide Web 3

Modellare conoscenza in OWL

- Il linguaggio prevede tre elementi:
  - **Entità**: elementi usati per riferirsi a oggetti del mondo reale. Sono elementi atomici che possono essere usati negli assiomi
  - **Assiomi**: affermazioni (statement) di base espressi da un'ontologia OWL
  - **Espressioni**: combinazioni di entità che costituiscono descrizioni complesse sulla base di altre

Modellare conoscenza in OWL

- Il linguaggio prevede tre elementi:
  - **Entità:**
    oggetti, categorie e relazioni che costituiscono gli statement
    Maria, donna, Luca, Anna, essere-sposati
  - **Assiomi**:
    Maria è una donna
    Luca e Anna sono sposati
  - **Espressioni**:
    definite tramite costruttori, supponiamo di avere la categoria medico e la categoria donna, è possibile combinarle definendo la categoria delle donne medico

File OWL

- **Preambolo**
  Prefix(:=<http://example.com/owl/families/>)
  Prefix(otherOnt:=<http://example.org/otherOntologies/families/>)
  Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
  Prefix(**owl:=<http://www.w3.org/2002/07/owl#>**)
  Ontology(<http://example.com/owl/families>
  …)
- I prefissi permettono di fare sinteticamente riferimento a elementi definiti in altre ontologie (o altri file)
- Il prefisso più l'etichetta sono composti nell'identificatore dell'elemento di interesse, esempio **owl:Thing** diventa **http://www.w3c.org/2002/07/owl#Thing**
- **Ontology specifica l'URI del file contenente l'ontologia definita.**
  Sarà generalmente salvato nel file system ed eventualmente accessibile via web

OWL: sintassi

- È possibile scrivere ontologie OWL utilizzando sintassi differenti:

  1) **Functional-Style (quella che useremo):**
     ClassAssertion(:Persona :Maria)
     noi useremo questa notazione per introdurre alcuni elementi sintattici importanti del linguaggio

  2) **RDF/XML (prima a essere utilizzata):**
     <Persona rdf:about="Maria"/>

  3) Turtle:
     :Maria rdf:type :Person .

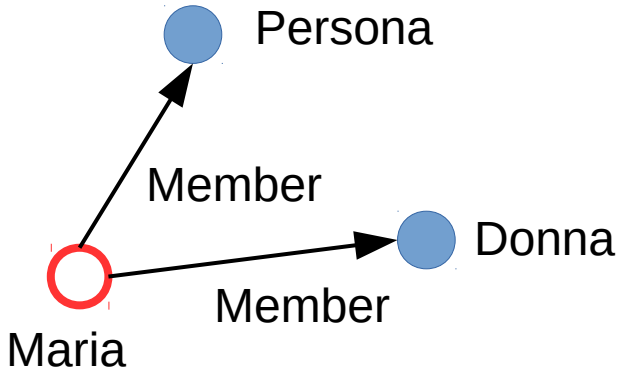
  4) Manchester:
     Individual: Maria

Elementi della KB

- **Nella terminologia di OWL:**
  - Oggetti (in FOL costanti): **individui**
  - Categorie (in FOL predicati unari): **classi**
  - Relazioni (in FOL predicati binari): **proprietà**
- **Esempi:**
  - **Dichiarazione di individuo (assioma):**
    `Declaration( NamedIndividual( :Maria ) )`
  - **Dichiarazione di classe (assioma):**
    `Declaration( Class( :Persona ) )`
  - **Dichiarazione di proprietà (assioma):**
    `Declaration( ObjectProperty( :donna ) )`

Classi e istanze

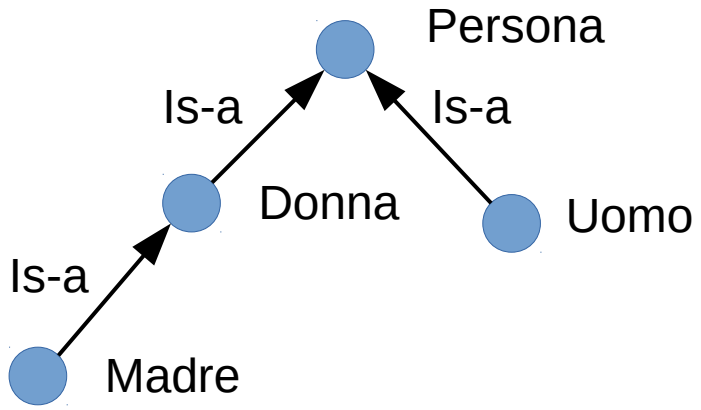
- **ClassAssertion** lega un' istanza a una classe, il primo argomento identifica la classe, il secondo l' individuo del quale si dichiara una proprietà
- **Esempi**:
  ClassAssertion(:Persona :Maria)
  ClassAssertion(:Donna :Maria)
  Maria è un individuo, membro (istanza) delle classi Persona e Donna



Relazione sottoclasse, equivalent, disjoint

- SubClassOf permette di definire tassonomie tramite la specificazione di relazioni di sottoclasse. Il primo parametro indica la sottoclasse, il secondo la sopraclasse. Tutte le istanze della sottoclasse apparterranno anche alla sopraclasse

- **Esempi**:
  SubClassOf(:Donna :Persona)
  SubClassOf(:Madre :Donna)
  Madre è sottoclasse di Donna, che è sottoclasse di Persona (axioms)

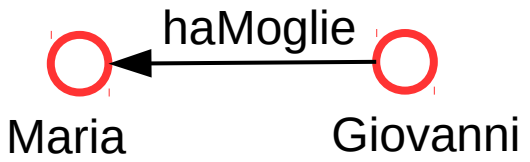


- **EquivalentClasses(:Persona :Umano)** indica l' equivalenza di due classi. È come dire che Persona è sottoclasse di Umano e al contempo Umano lo è di Persona (axiom)

- **DisjointClasses(:Donna :Uomo)** indica che le due classi non hanno istanze in comune (axiom)

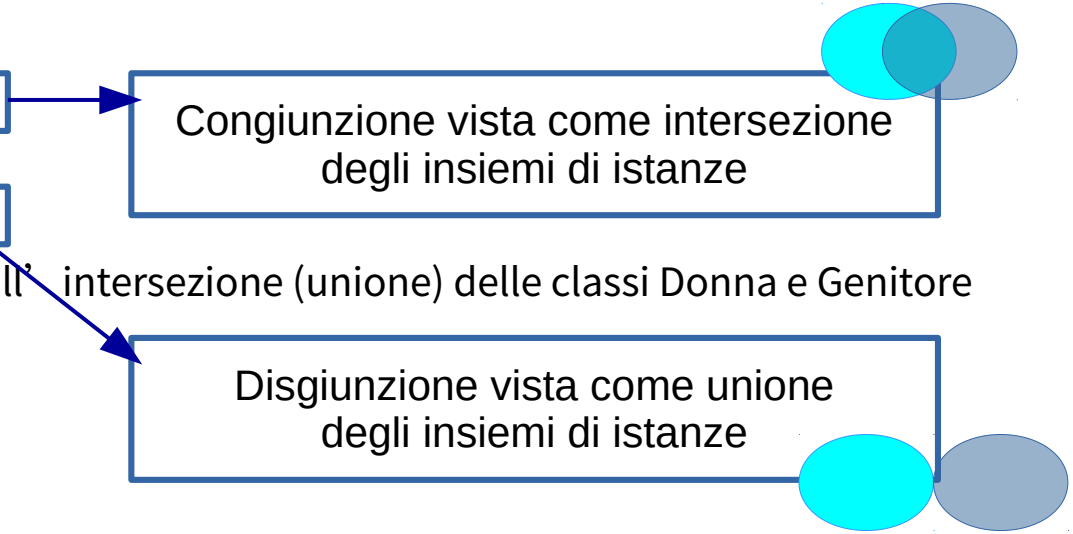
Proprietà

- ObjectPropertyAssertion permette di legare due individui tramite una proprietà
- **Esempio**
  ObjectPropertyAssertion(:haMoglie :Giovanni :Maria)
  l'individuo Giovanni ha per moglie l'individuo Maria (assioma)
- È possibile, se necessario, specificare che una proprietà è sottoproprietà di un'altra, esempio avere moglie è sottoproprietà di avere un coniuge (axiom):
  SubObjectPropertyOf(:haMoglie :haConiuge)
- È opzionalmente possibile caratterizzare dominio e codominio di una proprietà
- **Esempio**
  la proprietà haMoglie caratterizza le istanze di Uomo legandole a istanze di Donna
  ObjectPropertyDomain(:haMoglie :Uomo)
  ObjectPropertyRange(:haMoglie :Donna)



Classi complesse

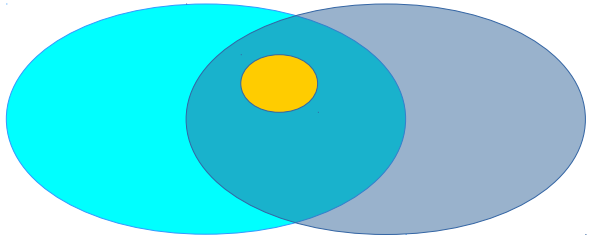
- È possibile specificare classi, e relazionarle, utilizzando opportuni costrutturi, esempi:
- EquivalentClasses(:Madre
  ObjectIntersectionOf(:Donna :Genitore))
  EquivalentClasses(:Genitore
  ObjectUnionOf(:Madre :Padre))
  la classe Madre (Genitore risp.) è equivalente all' intersezione (unione) delle classi Donna e Genitore (Madre e Padre)



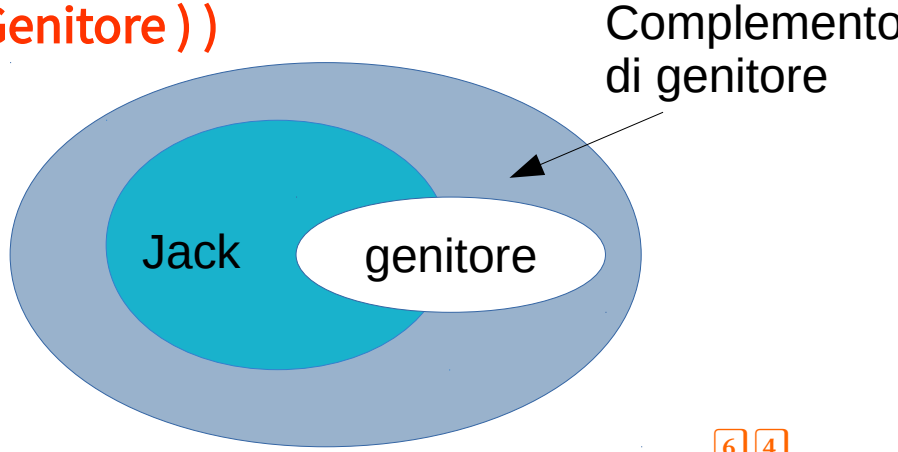
Intersection, Union … sono esempi di espressioni

Classi complesse

- SubClassOf(:Nonno
  ObjectIntersectionOf(:Uomo :Genitore))
  Nonno è sottoclasse dell' intersezione fra Uomo e Genitore



- ClassAssertion(
  ObjectIntersectionOf(:Persona
  ObjectComplementOf(:Genitore))
  :Jack)
  L' individuo Jack è una Persona non Genitore



Quantificazione esistenziale e universale

QUANTIFICAZIONE ESISTENZIALE

EquivalentClasses(
:Genitore
ObjectSomeValuesFrom(:haFiglio :Persona))

La classe Genitore è l' insieme di quegli individui che hanno almeno un' istanza di Persona che è loro figlio

QUANTIFICAZIONE UNIVERSALE

EquivalentClasses(
:PersonaFelice
ObjectAllValuesFrom(:haFiglio :PersonaFelice))

una persona è felice quando tutti i suoi figli sono felici. Include come felici anche tutte le persone che non hanno figli

Datatype

- I data type permettono di specificare insiemi di valori tramite i quali è possibile descrivere istanze e proprietà, per esempio:

  DataPropertyDomain(:haEta :Persona)
  DataPropertyRange(:haEta xsd:nonNegativeInteger)

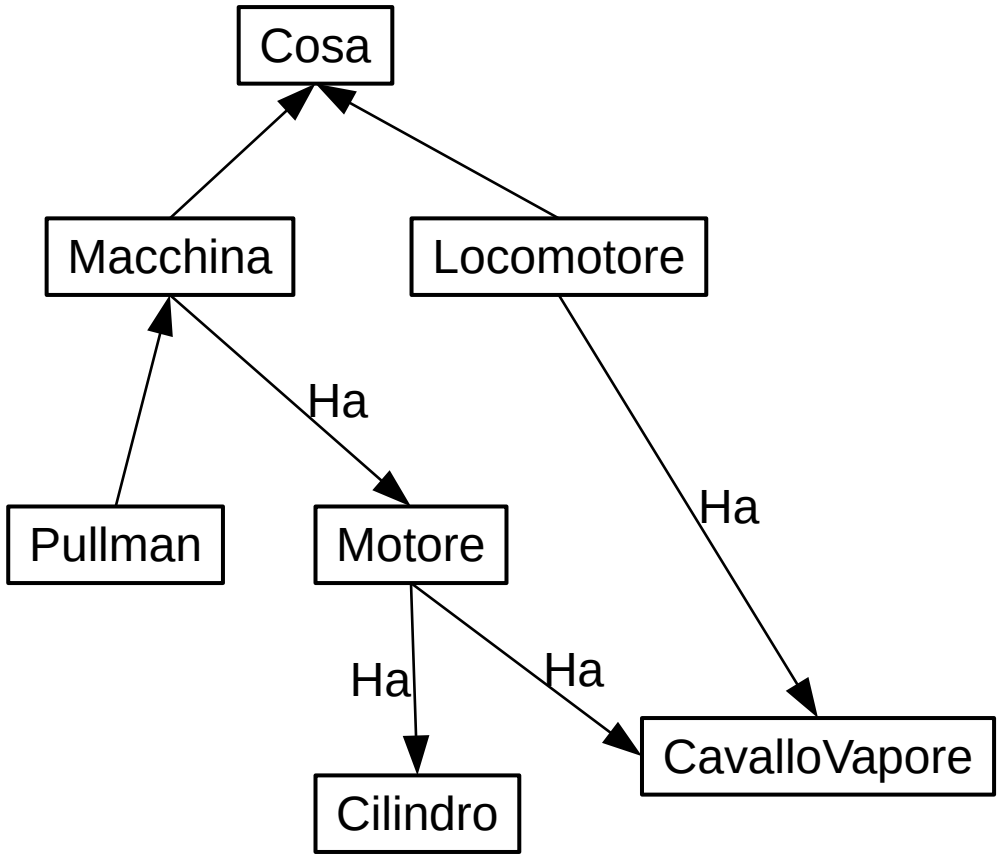
Allineamento ontologico

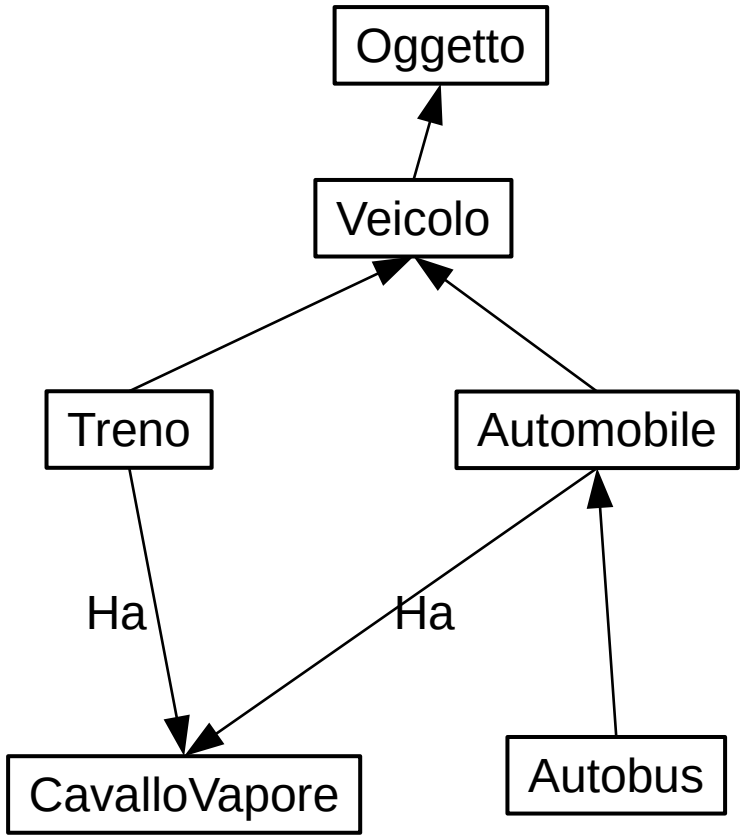
- Un problema frequente è combinare concettualizzazioni sviluppate separatamente e indipendentemente
- **Matching di ontologie**: date due ontologie O1 e O2 costruire un allineamento individuando le relazioni fra concetti corrispondenti
- La corrispondenza in generale sarà imperfetta

Esempio

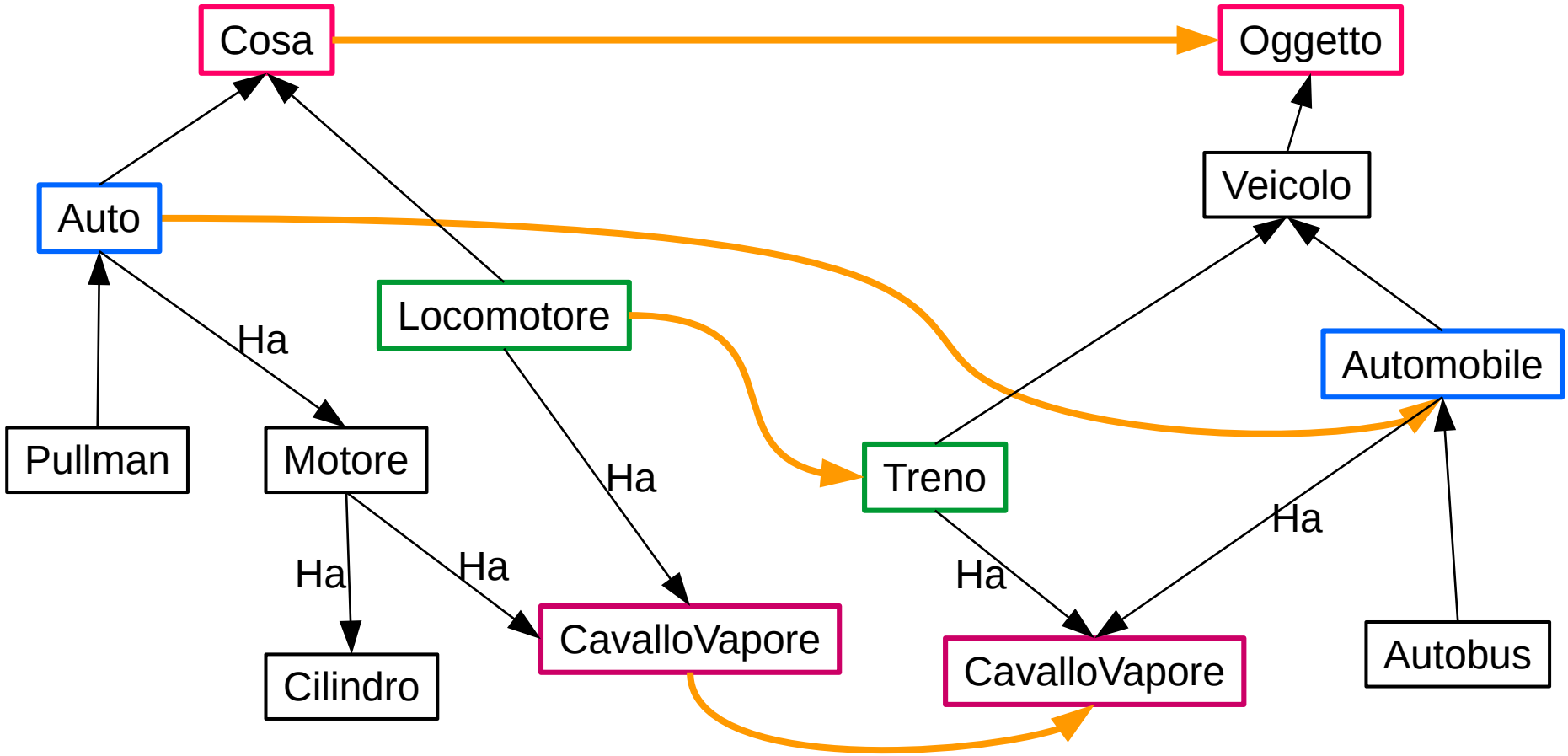
- Due agenti che interagiscono per scambio di messaggi fanno riferimento a **ontologie condivise**: (1) per riferisi al dominio del discorso; (2) per fare riferimento alle stesse definizioni di atti comunicativi e protocolli di interazione
- Fra le proposte **FIPA Ontology** (FIPA: Foundation for Physical Agents) postulano la presenza di un agente dedicato a gestire ontologie (ontology agent) e che fornisce fra i suoi servizi:
  - Discovery di ontologie pubbliche
  - Traduzione di espressioni in ontologie differenti
  - Rispondere a query relative alle differenze fra termini o ontologie

Allineamento ontologico, esempio





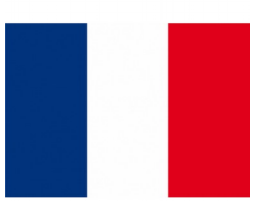
Allineamento ontologico, esempio



Relazioni fra ontologie

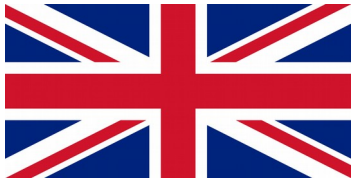
- **Identical**: O1 e O2 sono la stessa ontologia (esempio: un' ontologia e la sua copia in un disco mirror)
- **Equivalent**: condividono vocabolario e assiomatizzazione ma sono espresse in linguaggi differenti (esempio: SKOS e RDF)
- **Extension**: O1 estende O2 quando tutti i simboli definiti in O2 sono preservati in O1 insieme alle loro proprietà e relazioni ma non vale il viceversa
- **Weakly-Translatable**: siano Osource e Odest due ontologie, è possibile tradurre espressioni Osource in espressioni Odest con perdita di informazione
- **Strongly-Translatable**: slide
- **Approx-Translatable**: slide

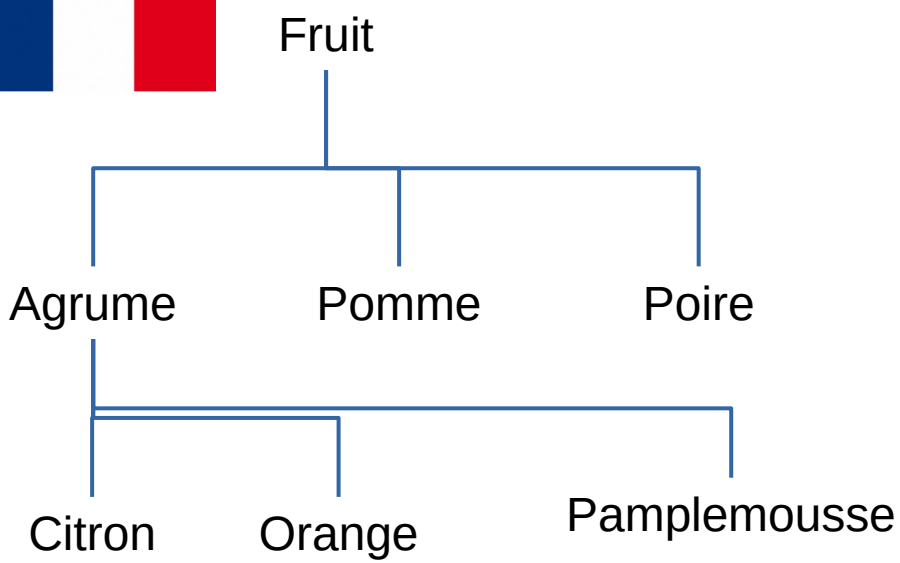
Weakly-Translatable, esempio

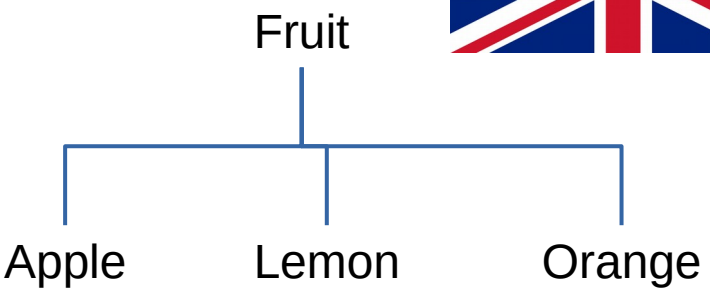


Osource

Odest







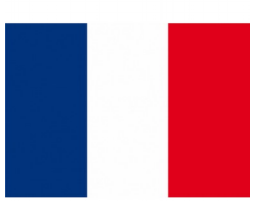
Osource è weakly-translatable in Odest usando il mapping:
Pomme → Apple, Citron → Lemon, Orange → Orange, Fruit → Fruit

Si ha perdita di informazione relativa a Poire, Pamplemousse e alla sottocategorizzazione di Agrume in Citron, Orange e Pamplemousse

Strongly-Translatable

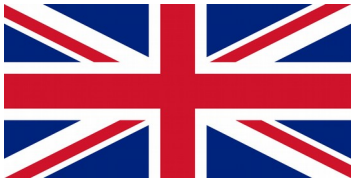
- Osource è strongly-translatable in Odest quando:
  - Il suo vocabolario è totalmente mappabile in quello di Odest
  - l'assiomatizzazione di Osource vale in Odest
  - Non c'è perdita di informazione
  - Non si introducono inconsistenze

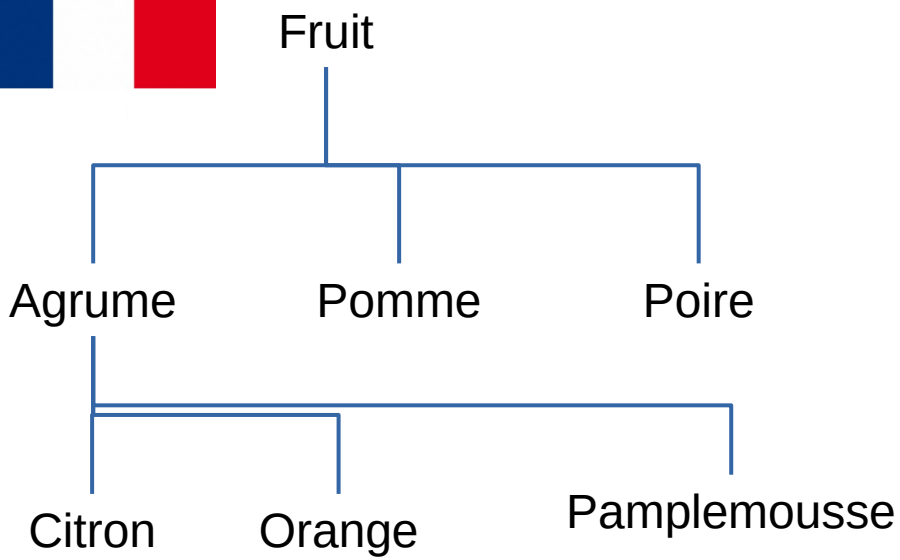
Strongly-Translatable, esempio

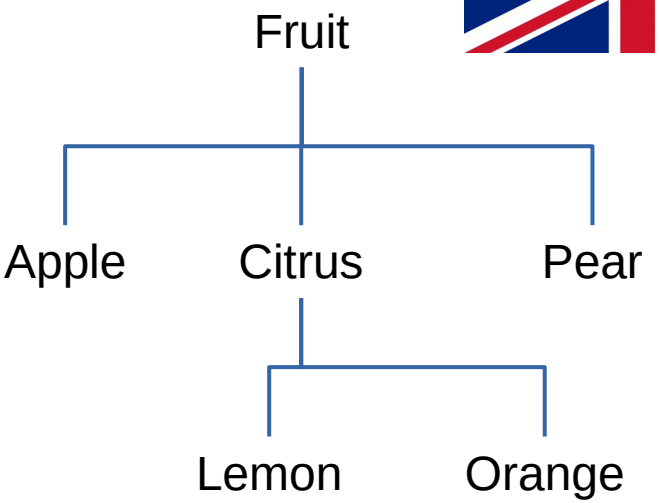


Odest

Osource







Osource è strongly-translatable in Odest usando il mapping:
Fruit → Fruit, Apple → Pomme, Pear → Poire, Citrus → Agrume, Lemon → Citron, Orange → Orange

Non si ha perdita di informazione. Si noti però che in Osource non vi è un concetto equivalente a Pamplemousse

Approx-Translatable

- Osource è **Approx-Translatable** in Odest quando è **Weakly-Translatable e possono essere introdotte delle inconsistenze**
- Tipicamente questo accade perché alcuni concetti che dovrebbero corrispondere l'uno con l'altro risultano affini ma non identici
- **Esempio**:
  Il coriandolo, a seconda della tradizione culinaria considerata, viene visto come affine al prezzemolo (là dove se ne usano le foglie) o al pepe (là dove se ne utilizzano i semi). È la stessa pianta ma a seconda dell'ontologia di riferimento potrà avere proprietà (ontologiche) differenti

Applicazioni

- **Beni culturali** (es. per risorse documentali http://www.sparontologies.net/ontologies)
- **Musei**
- **Applicazioni legali**
- **Applicazioni mediche** (es. Genoma, moltissime sono collezionate in https://bioportal.bioontology.org/ontologies)
- **Applicazioni geografiche**
- **Applicaizoni meteorologiche** (es. http://cmapspublic3.ihmc.us/rid=1040074543812_1046692348_15282/SupercellThunderstorms.cmap)
- **Curricula di studi transnazionali**
- …

OWL2 e DB: considerazioni

- I documenti OWL 2 conservano informazione ma nonostante ciò **OWL 2 non è un framework per basi di dati**
- Parte della terminologia evoca assonanze con i DB ma la **semantica sottostante è differente**:
  - un fatto non contenuto in un **DB** è considerato falso (**closed world assumption**) in **OWL 2** sarà mancante (**open world assumption**)
  - OWL **non richiede** che le uniche proprietà di un individuo siano quelle della classe a cui appartiene
  - **Classi e proprietà** possono avere **definizioni multiple**
  - Le informazioni che riguardano un individuo possono essere **distribuite in documenti diversi**

Ulteriori considerazioni

- OWL 2 non è un linguaggio di programmazione, è un **linguaggio di rappresentazione dichiarativo**
- Sulla conoscenza rappresentata in OWL 2 è possibile applicare dei programmi per effettuare del reasoning e rispondere a query
- **Is-a e Part-of non sono sufficienti per rappresentare qualsiasi tipo di conoscenza**

Rappresentare le azioni

"Le azioni non sono rappresentabili in termini di relazioni Is-a e Part-of"

Planning

- Molte applicazioni dei sistemi di inferenza concernono il **decidere quali azioni eseguire**, tipicamente per raggiungere un obiettivo
- **Pianificare** significa costruire una sequenza di azioni per soddisfare un certo fine
- Un **problema di pianificazione** include la specifica degli elementi di interesse del mondo, delle azioni a disposizione, degli obiettivi

Planning

- Il mondo è descritto da un insieme di variabili: tipicamente è espresso nel linguaggio **PDDL** (Planning Domain Definition Language)
- Uno stato è una **congiunzione di atomi ground**, in cui non compaiono funzioni, esempio: *At(Camion1, Bari) ∧ At(Camion2, Lecce)*
- La azioni sono descritte in maniera schematica e hanno un impatto limitato sul mondo
- In generale, in un certo stato solo un sottoinsieme delle azioni sarà applicabile e di queste solo una sarà applicata
- Se l' azione scelta viene applicata realmente subito nel mondo reale potrebbe non esserci possibilità di backtracking

Azioni – Situation Calculus

- La rappresentazione e il ragionamento su azioni si basa spesso sul **Situation Calculus**, una rappresentazione logica che identifica i concetti base di:
  - **Azione:** qualcosa che viene compiuto e influenza il mondo
  - **Situazione**: stati derivanti dall' esecuzione di qualche azione
  - **Fluente**: proprietà che può cambiare valore (fluire)
  - **Predicato atemporale** (eterno): sono funzioni o predicati il cui calcolo non è influenzato dalle azioni

Fluente

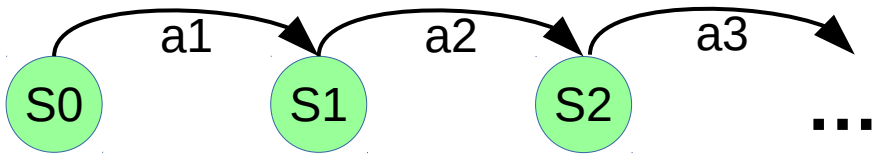
- Relazione o proprietà che può cambiare valore con l'esecuzione di azioni
- Viene specificata fra i suoi parametri la **situazione**, esempi:
  - **Adjacent(R1, R2, s)**: R1 e R2 sono adiacenti nella situazione s
  - **Holds(At(R, Loc), s)**: R si trova in posizione Loc nella situazione s

Azione

- Rappresenta qualcosa che viene compiuto
- In un contesto mono-agente non occorre indicare chi sia l'attore
  - **Esempio**: Move(R, L1, L2)
    rappresenta l'azione che sposta R da L1 a L2.
  - NB: In logica questo non è un predicato bensì è una funzione, restituisce un oggetto del dominio di riferimento
- Quindi un'azione è intesa come un oggetto intangibile, prodotto da una funzione (nell'esempio ternaria)

Legare situazioni e azioni

- Nel situation calculus il tempo non è espresso in modo esplicito ma è comunque scandito dalla sequenza degli eventi
- Si parte da una **situazione iniziale S0** e si applica una sequenza di **eventi** generando successivamente le **situazioni S1, S2**, ecc.



Legare situazioni e azioni

- Do(Azioni, S): funzione che restituisce la situazione raggiunta, applicando la sequenza di azioni indicate a partire dallo stato indicato:
  - **Do([], s) = s**
  - **Do([a | rimanenza], s) =**
    Do(rimanenza, Risultato(a, s))
- NOTA: due situazioni sono identiche esclusivamente se sono originate dallo stesso stato iniziale applicando la stessa sequenza di azioni. In altri termini una situazione è identificata dalla storia che l'ha prodotta

[Do(Azioni1, S1) = Do(Azioni2, S2)]
⇔ [(Azioni1=Azioni2) ∧ (S1=S2)]

Legare situazioni e azioni

- Do(Azioni, S): tramite questa funzione un agente può fare **proiezione**, cioè può ragionare sugli effetti delle azioni, in particolare può:
  - **verificare** se un corso d'azione attraversa **situazioni che godono di determinate proprietà**
  - **pianificare un corso d'azione**: quale sequenza di azioni permette di raggiungere **una situazione che gode di una specifica proprietà?**

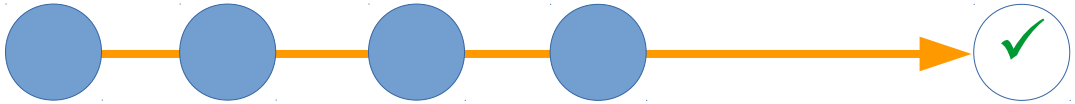
Ragionare su azioni: esempi

- **Proprietà che deve essere soddisfatta da tutte le situazioni attraversate:**

  Un trasportatore deve portare merci in diverse località. A ogni consegna può scegliere fra raggiungere la località successiva oppure fare rifornimento. Non deve mai rimanere a secco

- **Proprietà finale (goal):**

  Un ciclista deve comporre le diverse parti di una bicicletta per costruirla



Rappresentare le azioni

- Dobbiamo descrivere in logica anche le azioni
- Azione descritta da **ASSIOMI DI APPLICABILITÀ** e **ASSIOMI DI EFFETTO**:

Assioma di Applicabilità:

$\forall$ params, s Applicable(Action(params), s) $\Leftrightarrow$ Precond(params, s)

definisce che un' azione può (fisicamente) essere applicata in una situazione se e solo se valgono determinate precondizioni

- Applicable è un nuovo **predicato** che lega un' azione a una situazione
- Params è un **insieme di oggetti**
- Action(params) indica l' **applicazione dell' azione agli oggetti**
- Precond è una **formula** che rappresenta le precondizioni dell' azione

http://www.inf.ed.ac.uk/teaching/courses/plan/slides/Situation-Calculus-Slides.pdf

Esempio

- **ASSIOMA DI APPLICABILITÀ**
  $\forall$ params, s Applicable(Action(params), s) $\Leftrightarrow$ Precond(params, s)
- **Esempio**: Applicable(go(X,Y),S) $\Leftrightarrow$ At(X, S) $\wedge$ Adjacent(X, Y)
  - go(X, Y) : azione, andare dalla posizione X alla posizione Y
  - Params è l'insieme costituito dalle variabili X e Y
  - At(Agente, X, S) $\wedge$ Adjacent(X, Y) è la precondizione. Go(X, Y) può essere eseguita a patto che l'agente sia in X (fluente) e che X sia adiacente a Y (predicato atemporale)

http://www.inf.ed.ac.uk/teaching/courses/plan/slides/Situation-Calculus-Slides.pdf

Rappresentare le azioni

- **ASSIOMI DI EFFETTO**:

  $\forall$ params, s Applicable(Azione(params), s) $\Rightarrow$
  Effects(params, Result(Action(params), s))

  specifica gli effetti di un' azione sulla situazione in cui è eseguita

  - Effects è una **formula** vera nello stato risultante dall' esecuzione dell' azione
  - Result è una **funzione** che denota lo stato in cui si va eseguendo l' azione in s

http://www.inf.ed.ac.uk/teaching/courses/plan/slides/Situation-Calculus-Slides.pdf

Rappresentare le azioni

- **ASSIOMI DI EFFETTO**:

  $\forall$ params, s Applicable(Azione(params), s) $\Rightarrow$
  Effects(params, Result(Action(params), s))

  specifica gli effetti di un' azione sulla situazione in cui è eseguita

  - **Esempio**: Applicable(go(X, Y), S) $\Rightarrow$ At(Y, Result(go(X,Y),S))
  - L' effetto dell' azione applicabile go(X,Y) è che nella situazione risultante dall' esecuzione di tale azione in s (identificata da Result(go(X,Y),S)) l' agente (sottointeso nell' esempio in quanto unico) si trova alla posizione Y

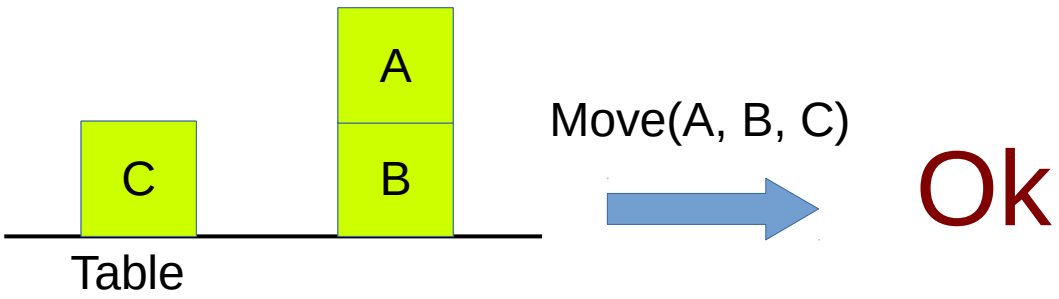
http://www.inf.ed.ac.uk/teaching/courses/plan/slides/Situation-Calculus-Slides.pdf

Esempio: mondo dei blocchi

- **Move(X, Y, Z)**: sposta X da Y a Z
- **Assioma di applicabilità:**
  - $\forall x,y,z,s$ Applicable(Move(x,y,z), s) $\Leftrightarrow$ Clear(x, s) $\wedge$ Clear(z, s) $\wedge$ On(x,y,s) $\wedge$ $x \neq z$ $\wedge$ $y \neq z$ $\wedge$ $x \neq$ Table

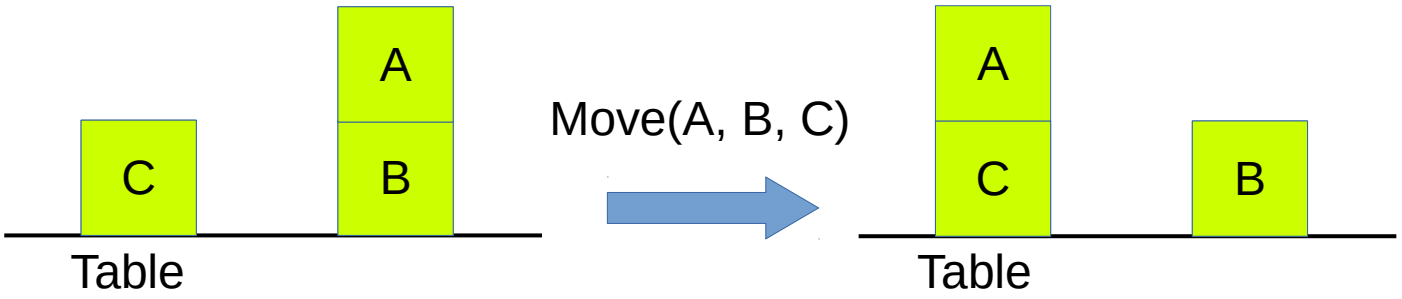
  quindi

- Move(A,B,A): no, non posso spostare un blocco sopra a se stesso
- Move(A,B,B): no, non denota uno spostamento



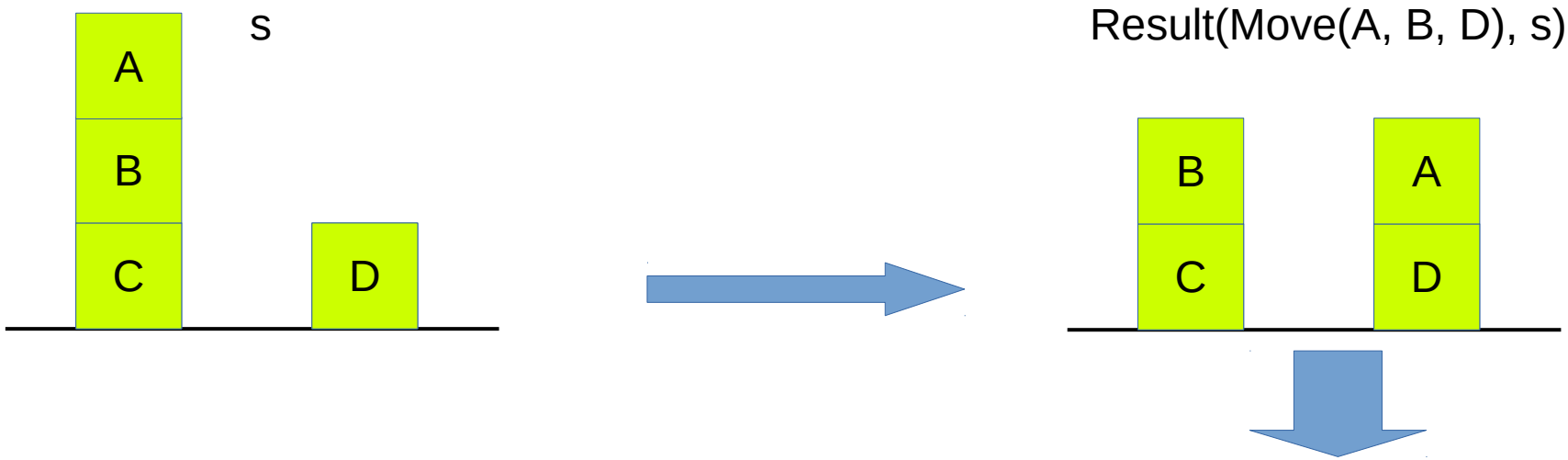
Esempio: mondo dei blocchi

- **Move(X, Y, Z)**: sposta X da Y a Z
- **Assioma di effetto:**
  - $\forall x,y,z,s$ Applicable(Move(x,y,z), s) $\Rightarrow$ On(x,z, Result(Move(x,y,z), s) $\wedge$ clear(y, Result(Move(x,y,z), s)



Inferenza

- Da (1) conoscenza di stato iniziale, (2) assiomi di applicabilità e (3) assiomi di effetto (KB nel seguito) è possibile derivare fatti

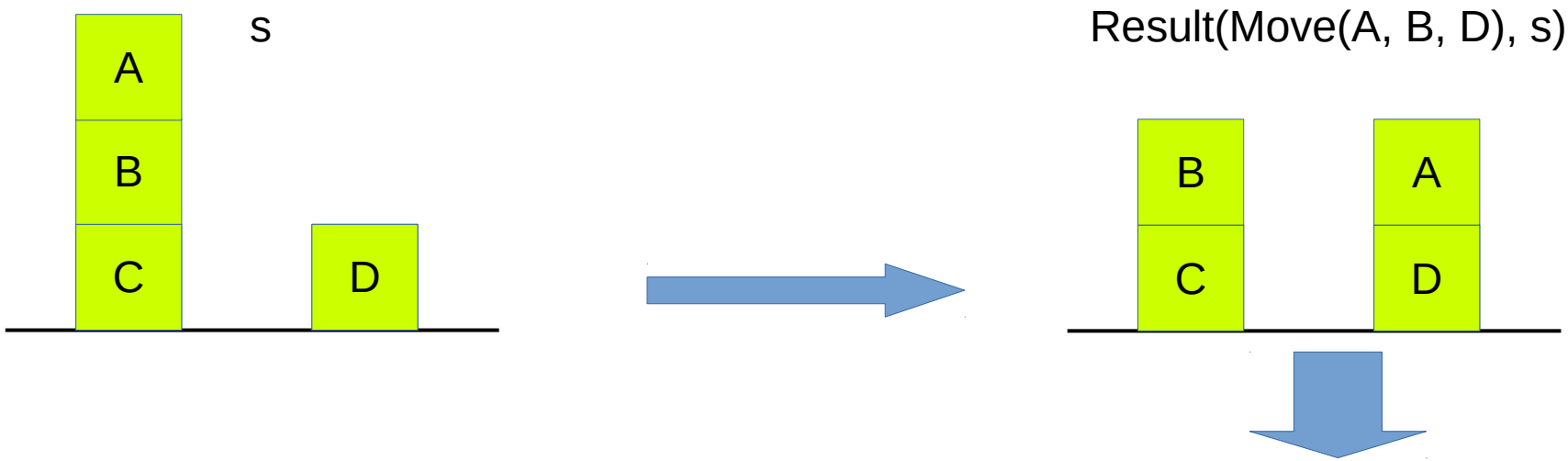


È possibile derivare che:
On(A, D, Result(Move(A, B, D), s))
Clear(B, Result(Move(A, B, D), s))

Per definizione di Move

Frame problem

- Dalla conoscenza di stato iniziale, assiomi di applicabilità e assiomi di effetto (KB nel seguito) **NON** è possibile derivare tutti i fatti che ci aspettiamo

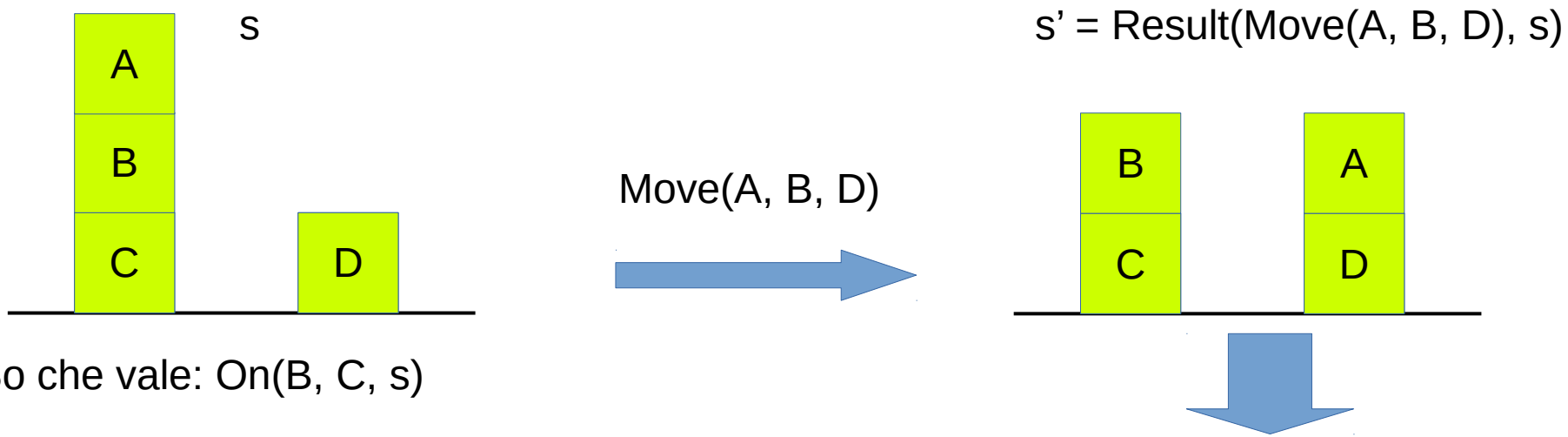


NON è possibile derivare che:
On(B, C, Result(Move(A, B, D), s))

Posso ragionare su cosa è cambiato ma non su cosa non è cambiato

Frame problem

- Dalla conoscenza di stato iniziale, assiomi di applicabilità e assiomi di effetto (KB nel seguito) **NON** è possibile derivare tutti i fatti che ci aspettiamo

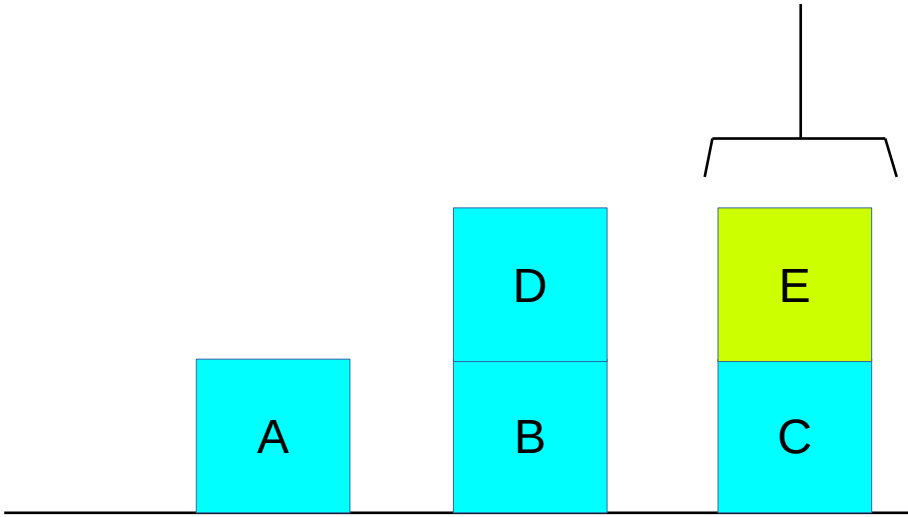


So che vale: On(B, C, s)

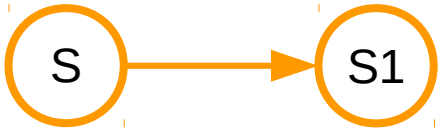
Abbiamo conoscenza su s ma senza assiomi che permettano di fare inferenza il sistema non può sapere se ciò che non è stato modificato da Move(…) vale anche in s'. Bisogna "dirglielo" in qualche modo.

Frame problem

- **Frame problem (problema della cornice)**:
  normalmente le azioni hanno un impatto limitato: come rappresentare ciò che non viene modificato da un' azione?



Quando il braccio afferra E l'unica cosa che cambia è che il braccio non è più vuoto. Per il resto la situazione rimane immutata



Il sistema automatico deve poter inferire cosa S1 eredita "as-is" da S

Frame problem 1: enumerare ciò che non cambia

- **Frame problem (problema della cornice)**:
  normalmente le azioni hanno un impatto limitato: come rappresentare ciò che non viene modificato da un' azione?

1) Rappresentazione esplicita tramite **assiomi di frame:**

$$\forall \text{params, vars, s } \text{fluent}(\text{vars}, s) \wedge \text{params} \neq \text{vars} \Rightarrow \text{fluent}(\text{vars}, \text{Result}(\text{Action}(\text{params}), s))$$

   Per ogni azione viene definito un assioma di questo tipo per ogni fluente

Frame problem 1: enumerare ciò che non cambia

- Rappresentazione esplicita tramite **assiomi di frame:**

  $\forall$ params, vars, s fluent(vars, s) $\wedge$ params $\neq$ vars $\Rightarrow$
  fluent(vars, Result(Action(params), s))

Esempio: azione: Move, vars = {x, y}, params = {z, w}

$\forall$ params, vars, s **on(x, y, s)** $\wedge$ x $\neq$ z $\Rightarrow$
on(x, y, Result(**Move(z, w)**, s))

$\forall$ params, vars, s **clear(x, s)** $\wedge$ x $\neq$ w $\Rightarrow$
clear(x, Result(**Move(z, w)**, s))

Frame problem

- **Problema**:
  occorre introdurre frame axioms per **tutti** i fluenti che non sono modificati da ciascuna azione!
- **Esempio**
  Se i blocchi ora possono essere dipinti di un colore, rivestiti di un materiale, ecc. occorrerà specificare frame axiom per ciascuna di queste proprietà:

$$\forall \text{params, vars, s colore}(x, y, s) \wedge x \neq z \Rightarrow \text{colore}(x, y, \text{Result}(\text{Move}(z, w), s))$$

$$\forall \text{params, vars, s materiale}(x, y, s) \wedge x \neq z \Rightarrow \text{materiale}(x, y, \text{Result}(\text{Move}(z, w), s))$$

...

Frame problem 2: evitare l'enumerazione

- introduciamo un modo per dire al sistema inferenziale che ciò che non è specificamente espresso come effetto è inteso rimanere immutato

Frame problem

- **ASSIOMA DI STATO SUCCESSORE:**

  Azione applicabile ⇒ (fluente vero nella situazione risultante
  ⇔(l'azione lo rende vero ∨
  era vero e l'azione non l'ha reso falso))

- Questo assioma esprime il modo in cui le **situazioni si evolvono** l'una dall'altra a seguito dell'esecuzione delle azioni. In particolare dice quali parti di una situazione sono ereditati dalla precedente

Unique action names

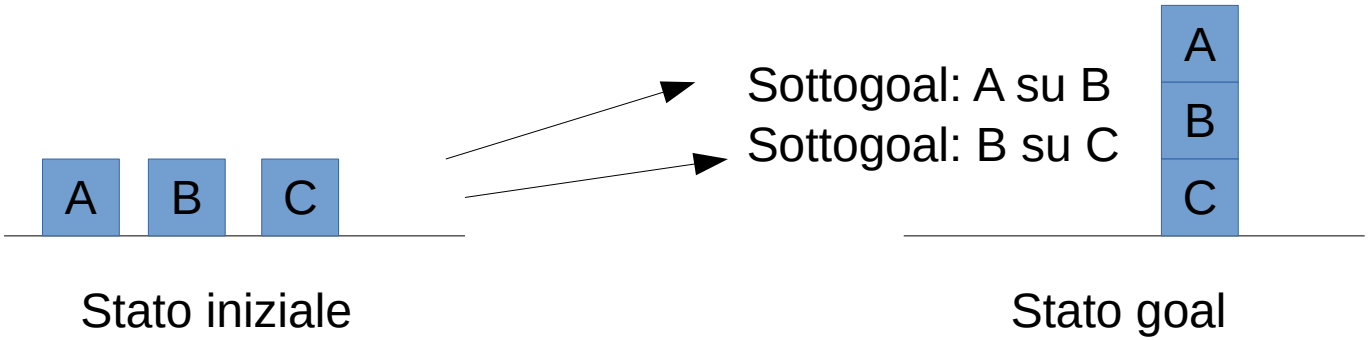
- Il situtation calculus è completato da assiomi che catturano:
  - che azioni con nomi diversi sono diverse:
    $A_i(x, \ldots) \neq A_j(y, \ldots)$
  - E per ogni nome di azione, che due usi di quel nome sono identici se e solo se gli argomenti sono identici:
    $A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \Rightarrow x_1=y_1 \wedge \ldots \wedge x_n=y_n$

Esempio

- Mondo dei blocchi (documentazione a parte)

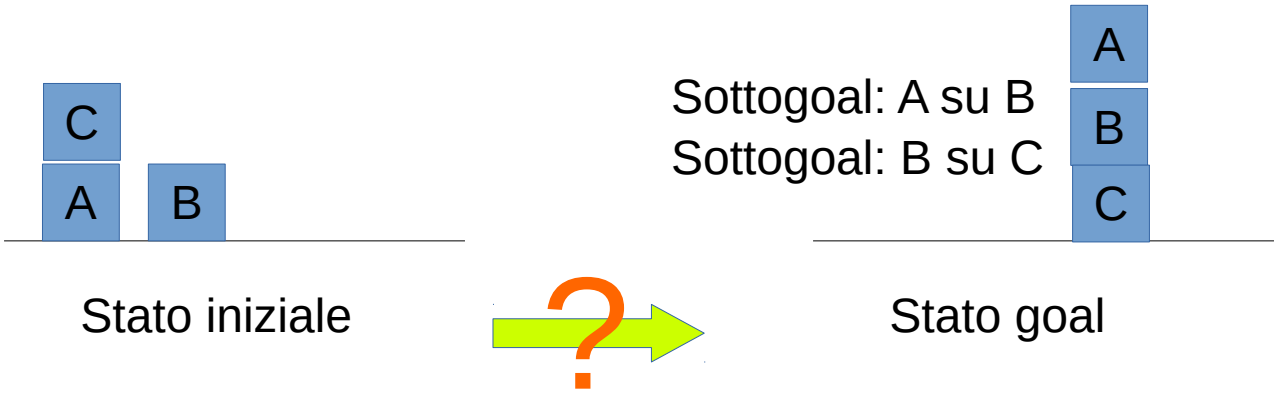
Perseguire goal complessi

- Un approccio tipico alla pianificazione consiste nello scomporre l'obiettivo in sottoobiettivi e conseguire questi ultimi uno per volta
- **Esempio**:



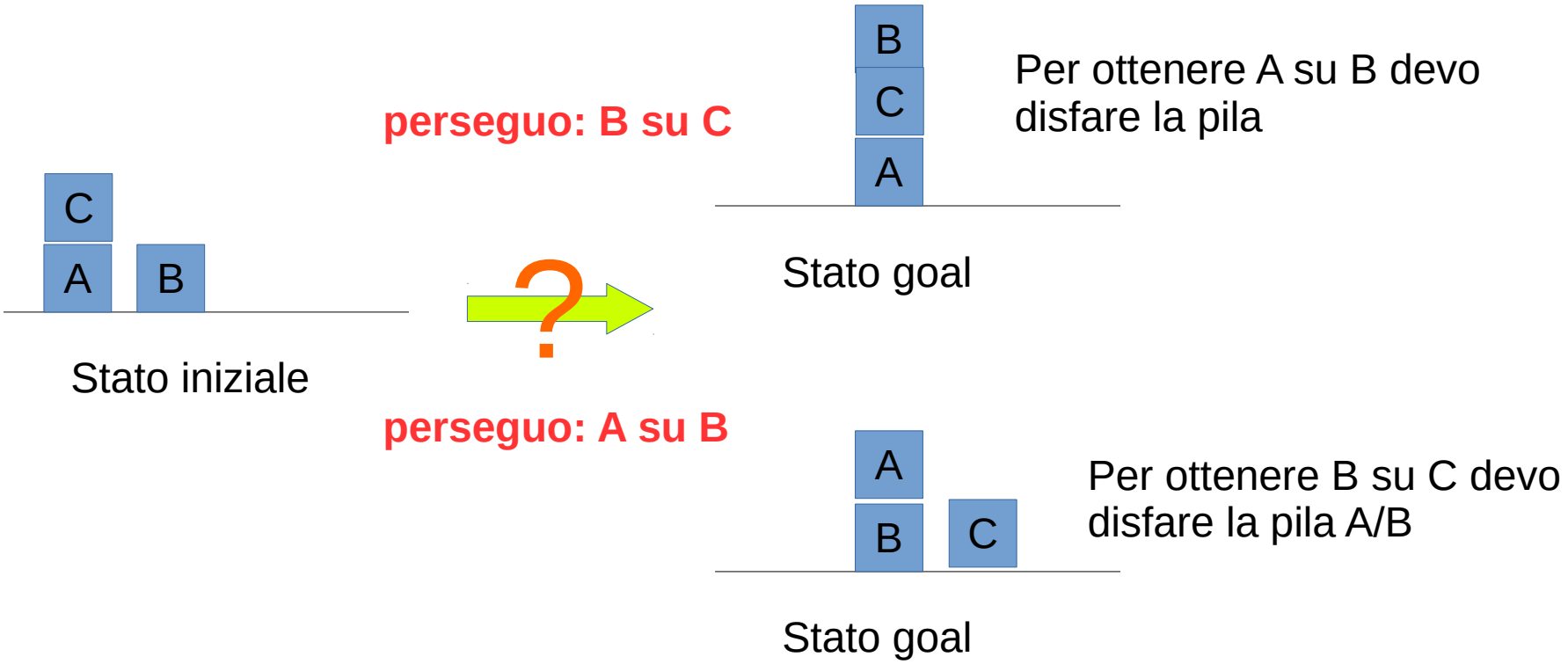
Anomalia di Sussman

- Non sempre il perseguimento degli obiettivi è sequenzializzabile, esempio:



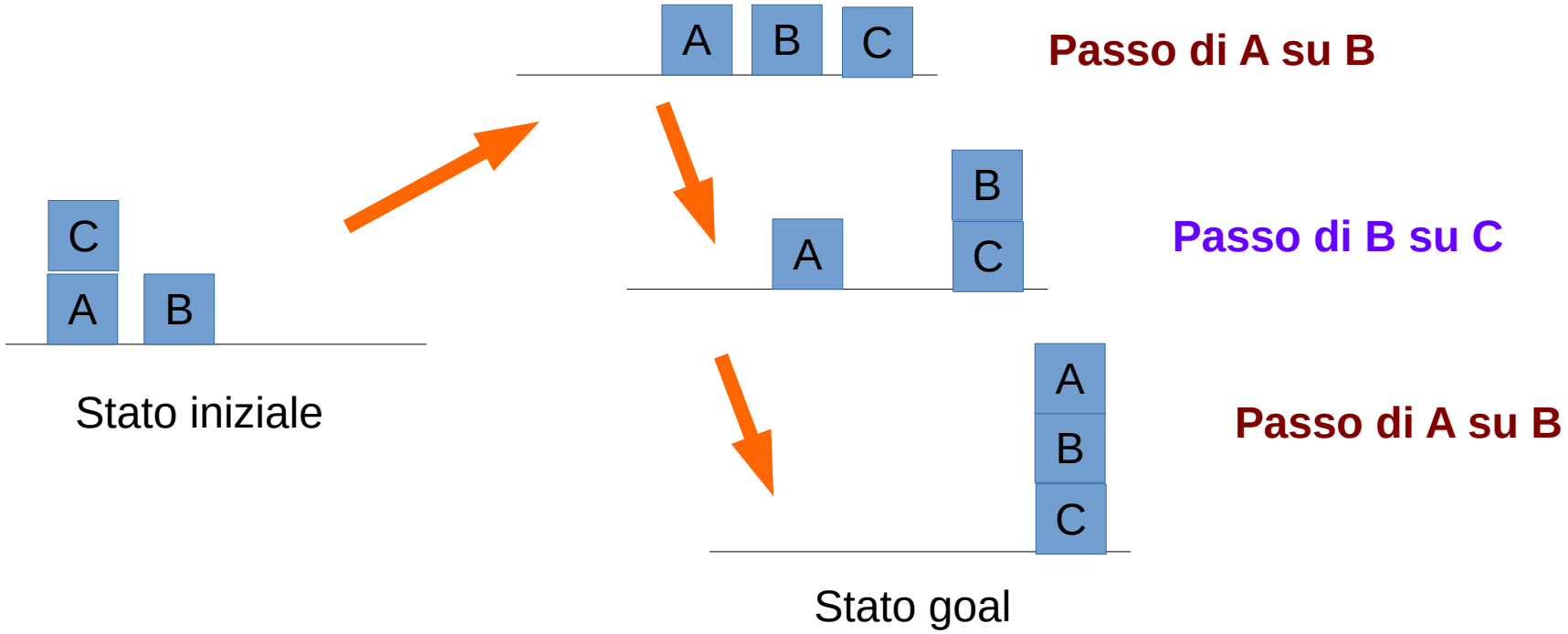
Anomalia di Sussman

- Non sempre il perseguimento degli obiettivi è sequenzializzabile, anzi alcune volte il perseguimento di un sottogoal può disfare passi effettuati per raggiungerne un altro. Esempio:



Interleaving dei passi

- Per risolvere efficientemente l’ esempio occorre fare interleaving dei passi di soluzione. Esempio:



Considerazioni

- Il situation calculus permette di usare FOL per problemi di pianificazione
- È stato fondamentale per definire il problema di pianificazione
- Nella pratica non è molto usato perché non esistono euristiche efficienti che guidino la ricerca della soluzione
- La pianificazione (planning) è parte del corso IA e Laboratorio della laurea magistrale in Intelligenza Artificiale e Sistemi Informatici