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

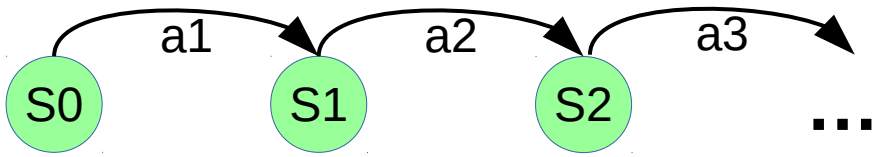
- Relazione o proprietà che può cambiare valore con l'esecuzione di azioni
- Viene specificata fra i suoi parametri la **situazione**, esempi:
  - **Adjacent(R1, R2, s)**: R1 e R2 sono adiacenti nella situazione s
  - **Holds(At(R, Loc), s)**: R si trova in posizione Loc nella situazione s

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?**

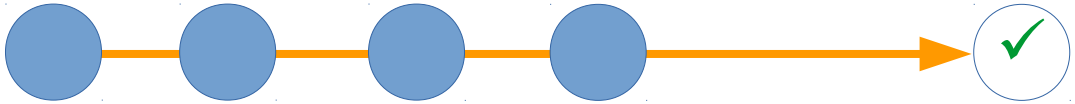
Ragionare su azioni: esempi

- **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

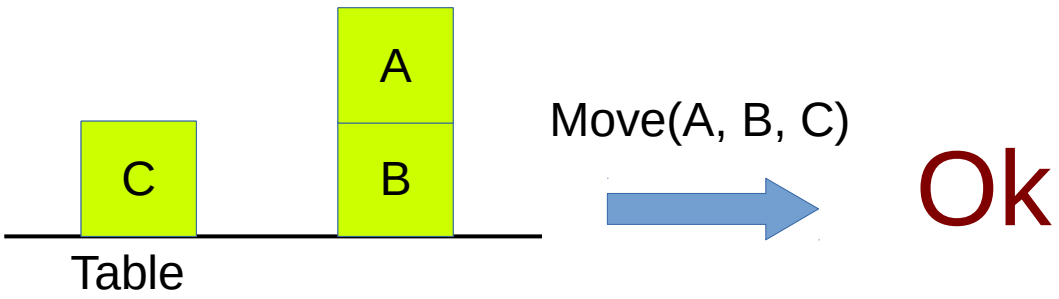
http://www.inf.ed.ac.uk/teaching/courses/plan/slides/Situation-Calculus-Slides.pdf

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

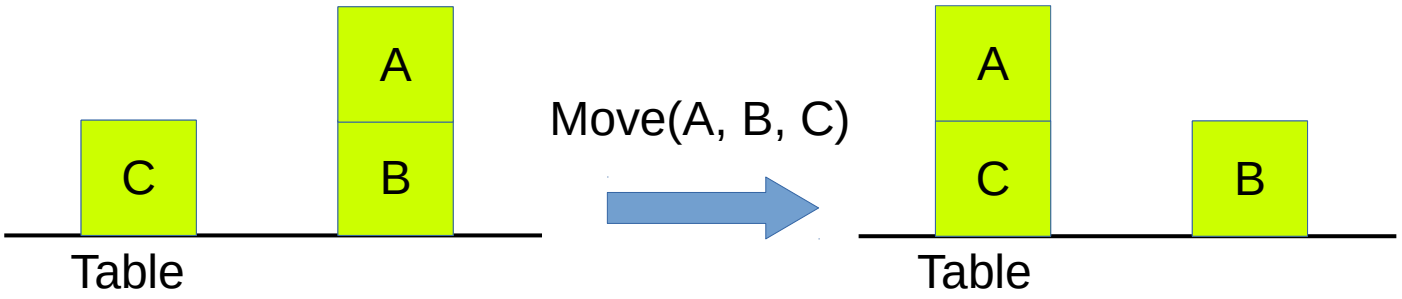
  quindi

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



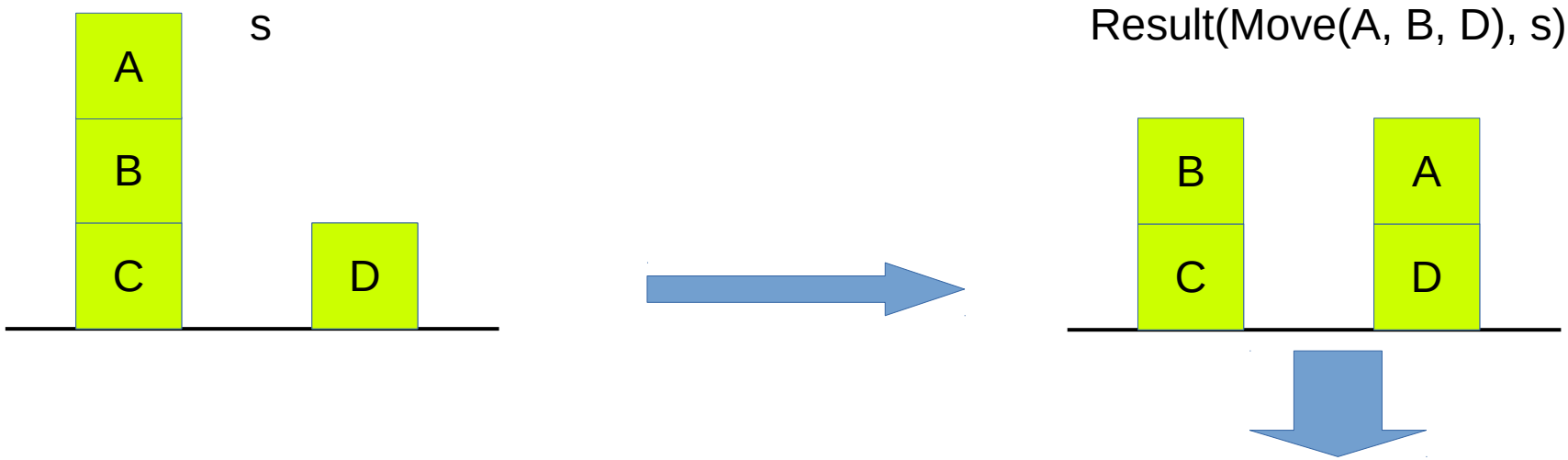
Esempio: mondo dei blocchi

- **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

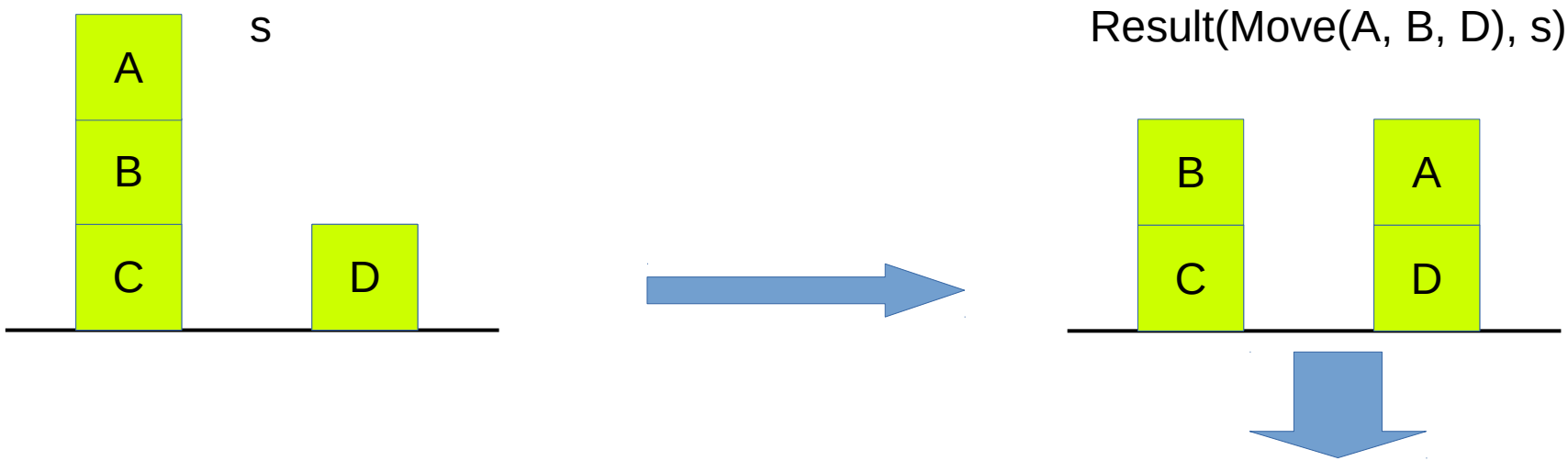


È possibile derivare che:
On(A, D, Result(Move(A, B, D), s))
Clear(B, Result(Move(A, B, D), s))

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

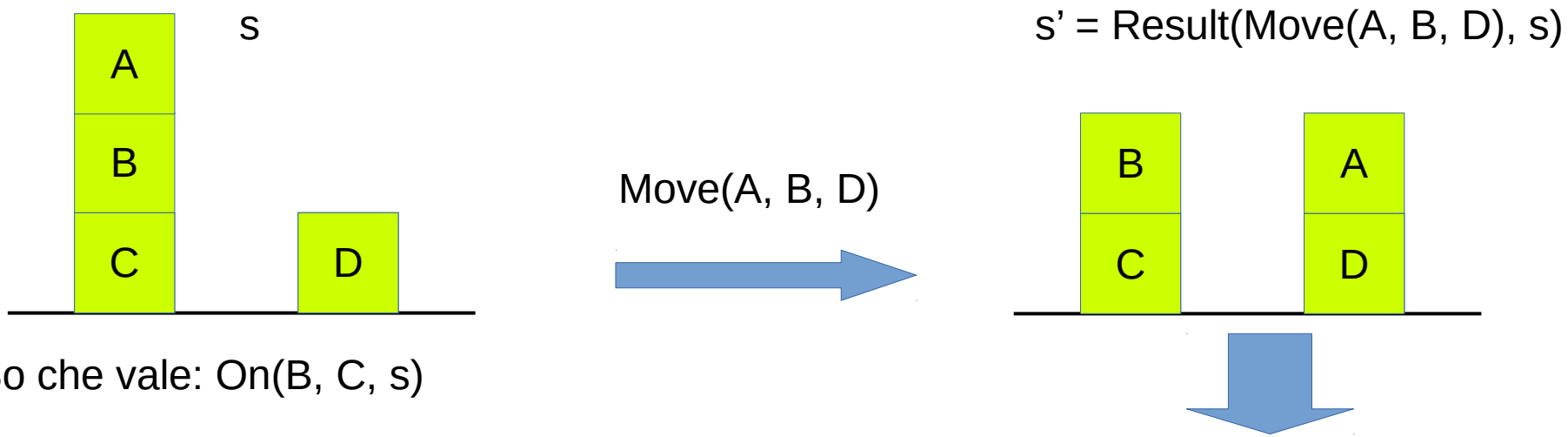


NON è possibile derivare che:
On(B, C, Result(Move(A, B, D), s))

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

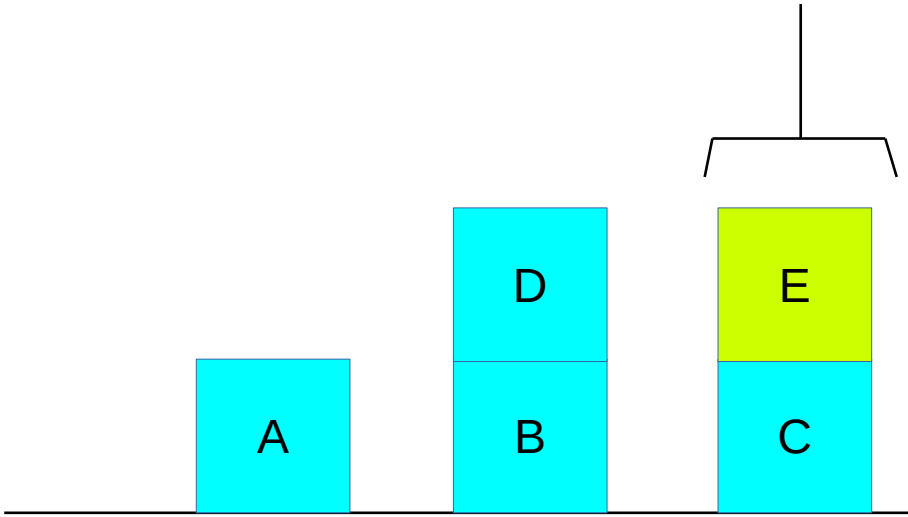


So che vale: On(B, C, s)

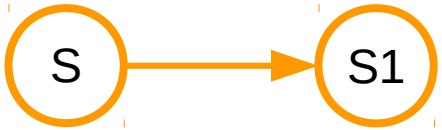
Abbiamo conoscenza su s ma senza assiomi che permettano di fare inferenza il sistema non può sapere se ciò che non è stato modificato da Move(…) vale anche in s'. Bisogna "dirglielo" in qualche modo.

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