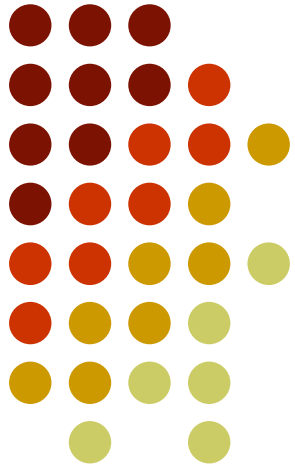
Representação de Conhecimento

Introdução à Inteligência Artificial

Profa. Josiane

David Poole, Alan Mackworth e Randy Goebel -
"*Computational Intelligence – A logical approach*" - cap. 5

julho/2007



Problemas e soluções

- Tipicamente o problema a ser resolvido ou a tarefa a ser feita também constitui uma solução e é definida informalmente
  - Entregue os pacotes prontamente quando eles chegarem
  - Determine o que está errado com o sistema elétrico da casa
- Para resolver um problema precisamos:
  - Expandir a tarefa e determinar o que constitui uma solução
  - Representar o problema
  - E somente então usar um computador para resolvê-lo

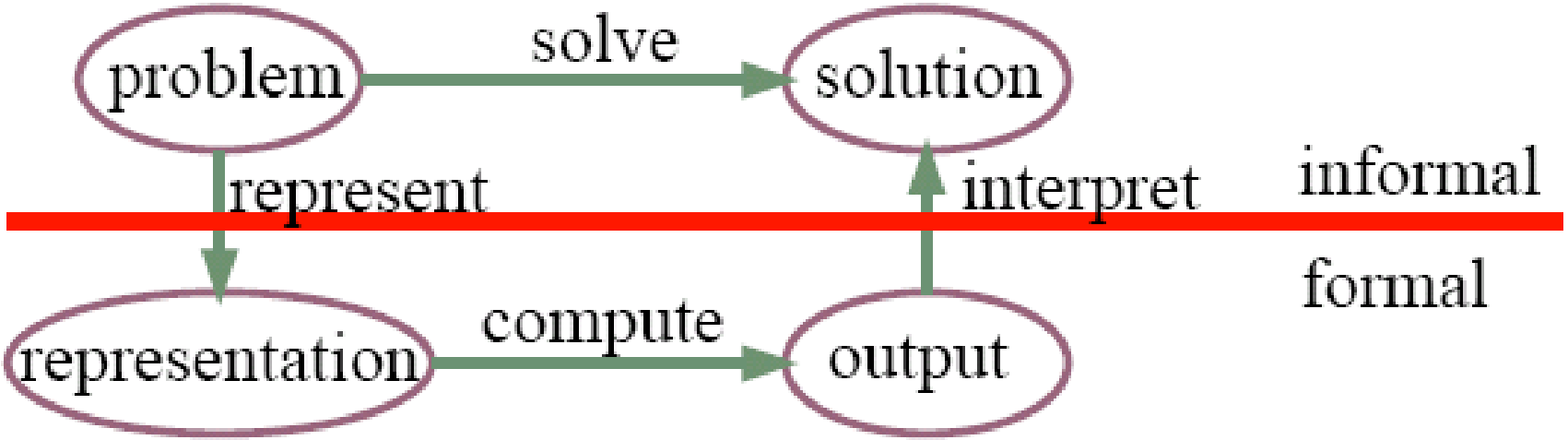
O que considerar para resolver um problema

- O que é uma solução para o problema?
- O que é necessário na linguagem para representar o problema?
- Como podemos mapear a descrição informal do problema para uma representação?
- Quais distinções do mundo são importantes para resolver o problema?
- Qual é o conhecimento necessário?
- Qual é o nível de detalhes necessário?

O que considerar para resolver um problema

- Quais estratégias de raciocínio são adequadas?
- O desempenho do pior caso e do caso médio são tempos críticos a minimizar?
- É importante que alguém entenda como a resposta foi derivada?
- Como podemos adquirir conhecimento? De especialistas? Ou de experiências?
- Como podemos debugar, manter e melhorar o conhecimento?

Framework de Representação de Conhecimento



Definindo uma solução

- Com uma definição informal de um problema, precisamos determinar o que constituirá uma solução
- Tipicamente problemas **não** são bem definidos
  - Muita coisa não é especificada, e o que não é especificado pode ser preenchido de qualquer forma
  - Quando ordenamos a um robô entregador que leve todo o material dispensável para o lixo, não esperamos que ele leve todas as coisas para lá
- Muito do trabalho em IA é motivado pelo **raciocínio de senso-comum**:
  - Queremos que o computador seja capaz de fazer conclusões de senso comum sobre suposições não-declaradas

Qualidade das soluções

- Qual é a importância se a resposta for errada ou alguma resposta estiver faltando?

- Classes de soluções:
  - **Solução Ótima**: a melhor solução de acordo com alguma medida de qualidade (utilidade)
  - **Solução Satisfatória**: uma que é boa o bastante de acordo com alguma descrição de quais soluções são adequadas
  - **Solução Aproximadamente Ótima**: uma para qual a medida de qualidade está muito perto daquela teoricamente possível
  - **Solução Provável**: uma que está apta a ser uma solução

Decisões e conseqüências

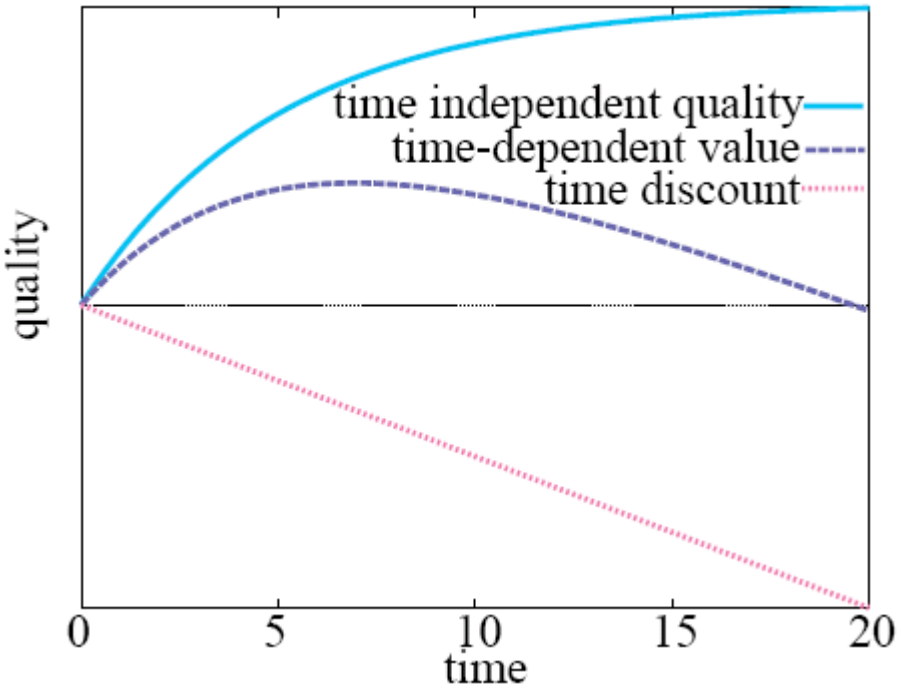
- Uma boa decisão é aquela que geralmente conduz a conseqüências boas.
  - Boas decisões podem ter conseqüências ruins. Decisões ruins podem ter conseqüências boas.
  - Um agente necessita de uma informação e decide acessar uma fonte de conhecimento
    - A informação está lá – conseqüência boa
    - A informação não está lá – conseqüência ruim

Disponibilidade da informação e qualidade da solução

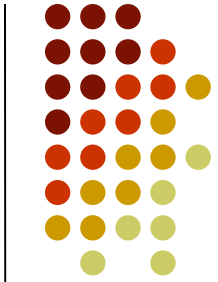
- Atividades dos agentes tem custo associado - determinar custo x benefício
- Nem toda a informação necessária para o agente está diretamente disponível para ele
  - Ações necessárias para obter informações tem custo associado
- Informação pode ser valiosa porque conduz a decisões melhores: **valor da informação**
- Quando definir uma solução temos que nos preocupar com:
  - Qual informação é necessária? Como ela pode ser obtida? Custo x benefício associado com as ações de obtê-las?

Custo da computação e qualidade da solução

- Uma boa solução pode depender do tempo de inferência
  - Ex: A decisão de atravessar a rua deve ser rápida
- **Algoritmo a qualquer tempo**: um algoritmo para o qual a qualidade da solução melhora com o tempo
- Compromisso entre encontrar a melhor solução e encontrar uma resposta rapidamente

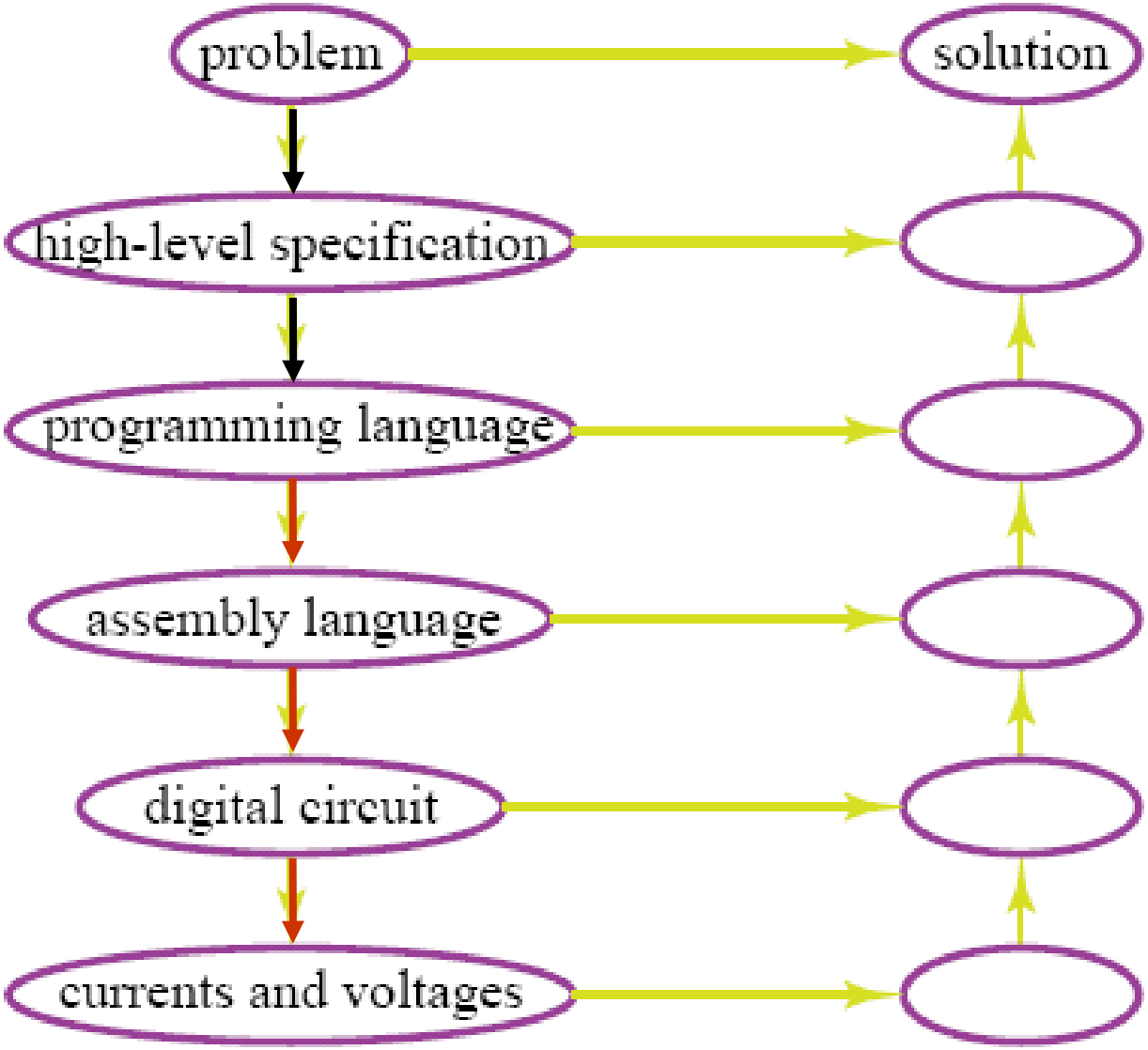


Escolhendo uma linguagem de representação

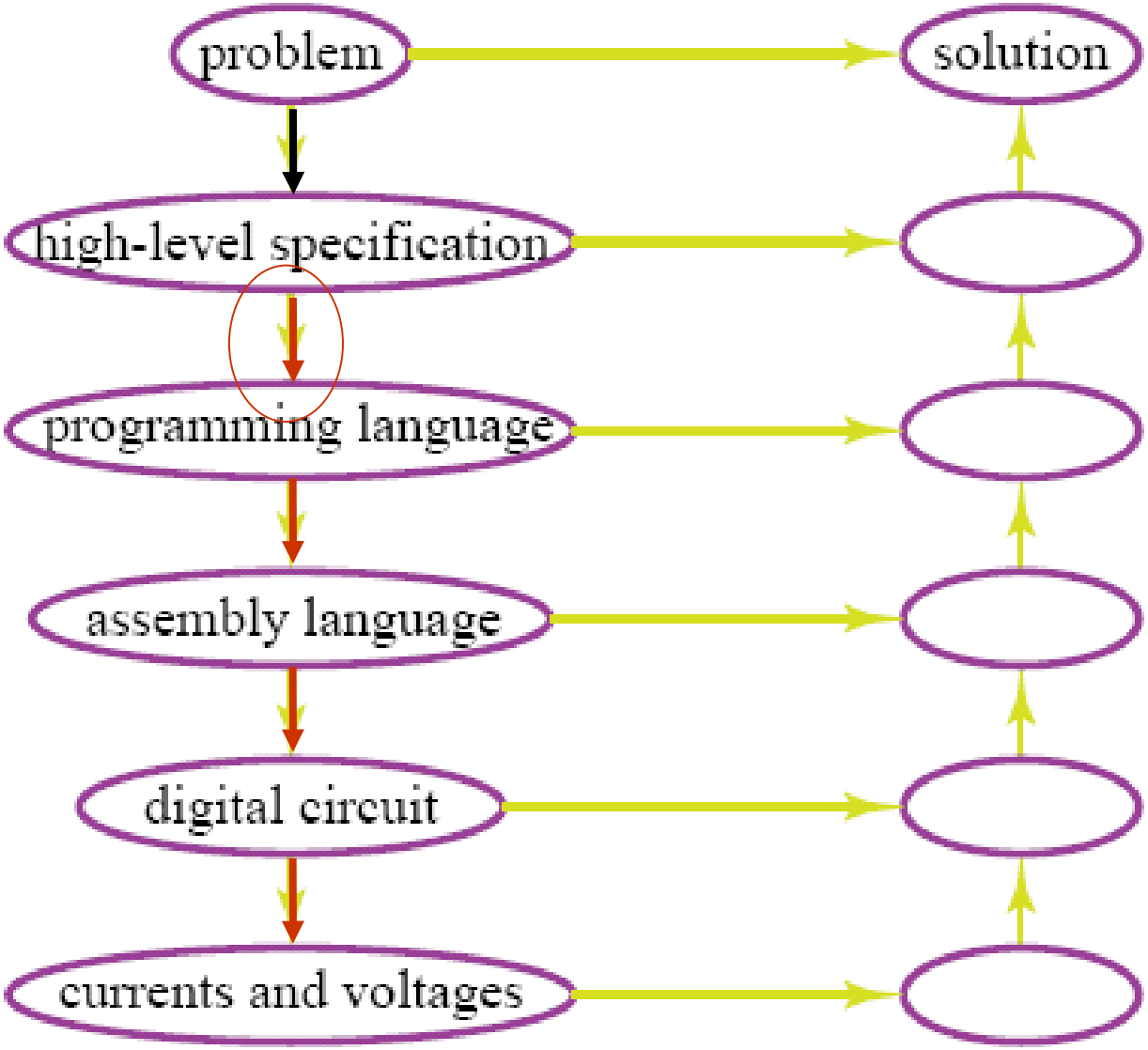


- Necessitamos representar um problema para resolvê-lo no computador
- Problema
  - Especificação do problema
  - Computação apropriada
- Exemplos de representação: C++, CILog/Prolog, ling. natural
  - Mapeamento (problema ⇔ representação), ambigüidade
- Uma **lógica** é uma linguagem + a especificação do que segue de um conjunto de sentenças da linguagem

Hierarquia das representações



Hierarquia das representações



Níveis de abstração

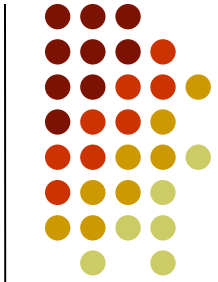
- **Nível de Conhecimento**: em termos do que o agente sabe e quais são os objetivos dele (sobre o mundo externo)
  - Não é especificado como a solução será computada
  - Nem mesmo quais das muitas estratégias possíveis disponíveis o agente irá utilizar

- **Nível de Símbolo**: em termos de quais símbolos o agente irá manipular
  - Quais símbolos o agente usa para implementar o nível de conhecimento

Mapeando o problema para a representação

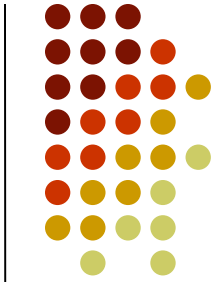
- Qual é o nível de abstração do problema que queremos representar?
- Quais objetos e relações no mundo que queremos representar?
- Com podemos representar o conhecimento para assegurar que a representação é natural, modular e fácil de manter?
- Como podemos adquirir informações de dados, sensores, experiências ou outros agentes?

Escolhendo um nível de abstração



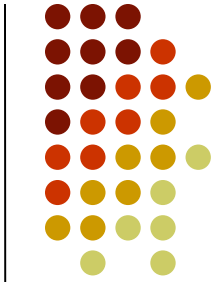
- Uma descrição de alto-nível é mais fácil para um ser humana especificar e entender
- Uma descrição de baixo-nível pode ser mais precisa e preditivo
  - Descrições de alto nível extraem detalhes que podem ser importantes para resolver o problema
- Quanto mais baixo o nível, maior a dificuldade de raciocinar com ele
- Podemos não conhecer a informação necessária para uma representação de baixo-nível
- Algumas vezes é possível utilizar múltiplos níveis de abstração

Escolhendo objetos e relações



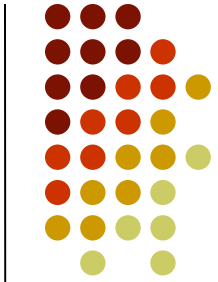
- Ex: vamos supor que “vermelho” é uma categoria apropriada para classificar objetos
  - Podemos tratar “*vermelho*” como uma relação unária e escrever que um pacote *a* é vermelho: ***vermelho*(*a*).**
    - Podemos perguntar o que é vermelho?: ?*vermelho*(*X*).
    - Mas é difícil perguntar qual é a cor do pacote *a?: <u>não</u> podemos perguntar ?X(a).*
  - *Nomes de predicados podem ser variáveis. Vermelho não precisa ser um predicado. Podemos considerar cores como indivíduos*
    - *Vermelho é uma constante, podemos usar o predicado cor(Obj, Val), para significar que o objeto Obj tem a cor Val*
    - *“O pacote a é vermelho”:* ***cor(a, vermelho).***
    - *Qual é a cor do pacote a?: cor(a, C).*

Escolhendo objetos e relações



- Com a representação *cor(Obj, Val), não conseguimos perguntar "Qual propriedade do pacote a tem valor vermelho?"*
- *Podemos fazer para toda relação o que fizemos com cor?*
  - *Podemos ver a relação cor como um indivíduo e escrever: prop(a,cor, red)*
  - *Agora podemos responder todas as perguntas*
- *A representação objeto-atributo-valor prop(Obj, Att, Val) significa que o objeto Obj tem o valor Val para o atributo Att*
  - *O domínio de um atributo é o conjunto de valores que o atributo pode assumir*

Escolhendo objetos e relações



- Para transformar "*a* é um pacote" (*pacote*(*a*)) na representação objeto-atributo-valor
  - *prop*(*a*, é_*um*, *pacote*), onde *é_um* é um atributo especial
  - *prop*(*a*, *pacote, true*), onde *pacote* é um atributo booleano

Escolhendo objetos e relações

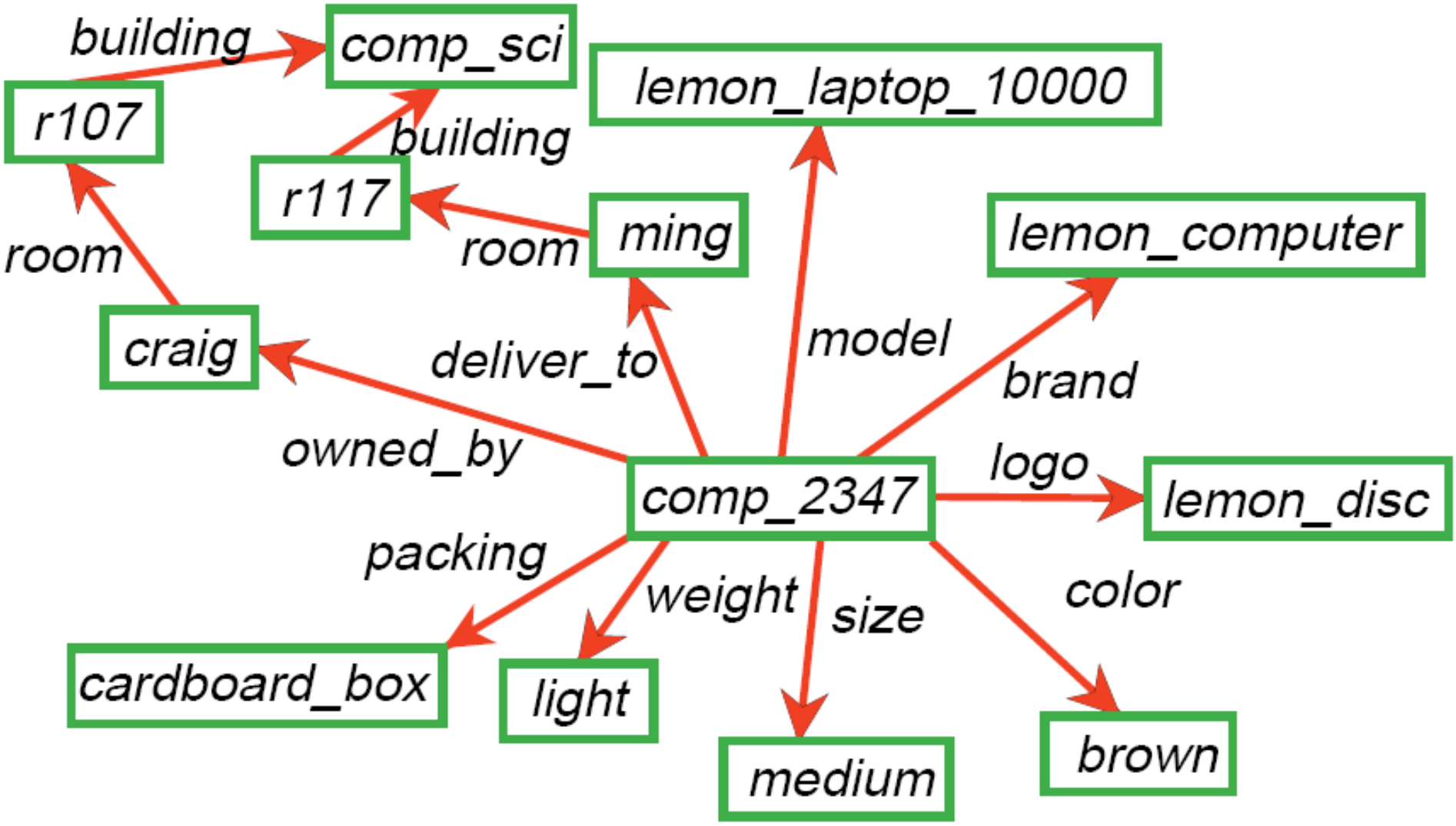
- Para representar a relação *programado*(*C, S, T, R*), onde S é uma seção, C é um curso, T é a hora e R é a sala.
  - "seção 2 do curso cs422 está programada para às 10:30 na sala cc208": *programado*(*cs422, 2, 1030, cc208*).
  - Para representar da forma objeto-atributo-valor, precisamos inventar novos indivíduos
  - Podemos chamar *reserva que tem um número de propriedades como curso, seção, hora e sala*
    - *Podemos nomear a reserva da sala como uma constante, b123:*
      - *prop(b123, curso, cs422).*
      - *prop(b123, seção, 2).*
      - *prop(b123, hora, 1030).*
      - *prop(b123, sala, cc208).*
    - *Uma representação modular e fácil de adicionar um novo atributo*

Redes Semânticas

- Quando temos somente uma relação, podemos omiti-la sem qualquer perca de informação
- Assim podemos representar uma relação *prop*(*Obj, Att, Val*) como um grafo onde *Obj* e *Val* são nós e *Att* é o rótulo do arco
- Estes grafos são chamados de **Redes semânticas**

Um exemplo de uma rede semântica

Informações sobre um computador



Programa equivalente em lógica
Informações sobre um computador

$prop(comp\_2347, owned\_by, craig).$

$prop(comp\_2347, deliver\_to, ming).$

$prop(comp\_2347, model, lemon\_laptop\_10000).$

$prop(comp\_2347, brand, lemon\_computer).$

$prop(comp\_2347, logo, lemon\_disc).$

$prop(comp\_2347, color, brown).$

$prop(craig, room, r107).$

$prop(r107, building, comp\_sci).$

$\vdots$

A notação gráfica

- Vantagens:
  - Os humanos entendem os relacionamentos com facilidade sem necessariamente saber a sintaxe
  - Ajuda os construtores da BC a estruturar seu conhecimento
  - Não temos que rotular nós os quais os nomes não fazem muito sentido (Ex: *b123*)

- Podemos agrupar todos os pares atributos-valores juntos em um simples objeto, isto é um ***frame***
  - É útil para trazer todas as informações sobre um objeto juntas
  - Não é necessário criar constantes artificiais
  - Atributos são chamados ***slots*** e valores são chamados ***fillers***

Frames

- Um exemplo de um frame para um computador

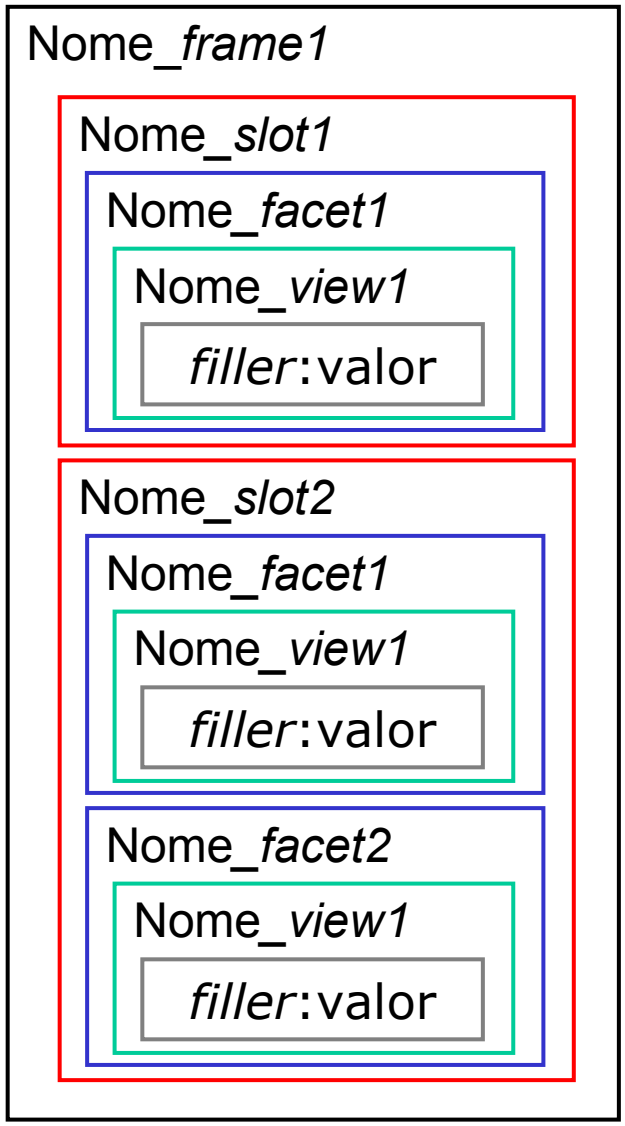
$$[owned\_by = craig,$$
$$deliver\_to = ming,$$
$$model = lemon\_laptop\_10000,$$
$$brand = lemon\_computer,$$
$$logo = lemon\_disc,$$
$$color = brown,$$
$$\ldots]$$

- Podemos definir uma *template* de um *frame* para cada tipo de indivíduo
  - Especifica quais slots são necessários e quais são opcionais para um indivíduo daquele tipo
  - Ajuda a construir a BC

Um Frame pode ter vários níveis

- É formado por um conjunto de ***slots***, que representam atributos dos objetos
- *Slots* podem possuir várias ***facets***, que trazem informações particulares de cada *slot* (atributo)
  - Exemplo: faixa de valores possíveis para o *slot*, ou forma de calculá-los
- Uma *facet* pode possuir várias ***views***, que representam diferentes contextos em que aquela *facet* pode ser avaliada
- Cada *view* possui um ***filler***, ou seja o valor daquela *view*, para aquela *facet*, para aquele *slot*
- Podemos ter *deamons* (procedimentos) associados a *slots*
  - Ex: se o valor de determinada *facet* é x, crie a *facet* y e z.

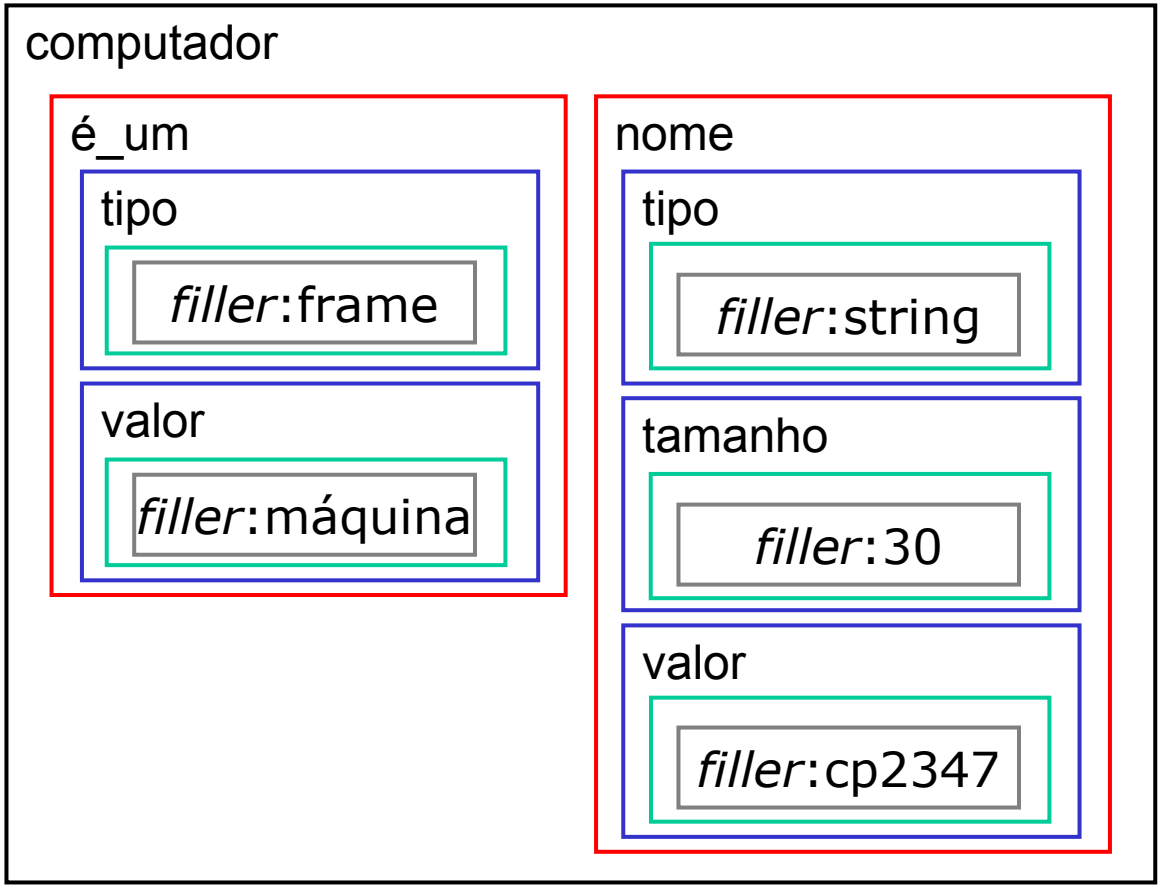
Exemplo: um Frame de vários níveis




```
Frame1[
   slot1[
      facet1[ view1[ valor]]
   ],
   slot2[
      facet1[ view1[ valor]],

      facet2[ view1[ valor]]
   ]
]
```

Uma visão gráfica de um frame de vários níveis



Um frame de vários níveis representado como listas em Prolog

```
computador[
   é_um[
      tipo[ [ frame]],
      valor[ [ máquina]]
   ]
   nome[
      tipo[ [ string]],
      tamanho[ [ 30]],
      valor[ [ cp2347]]
   ]
]
```

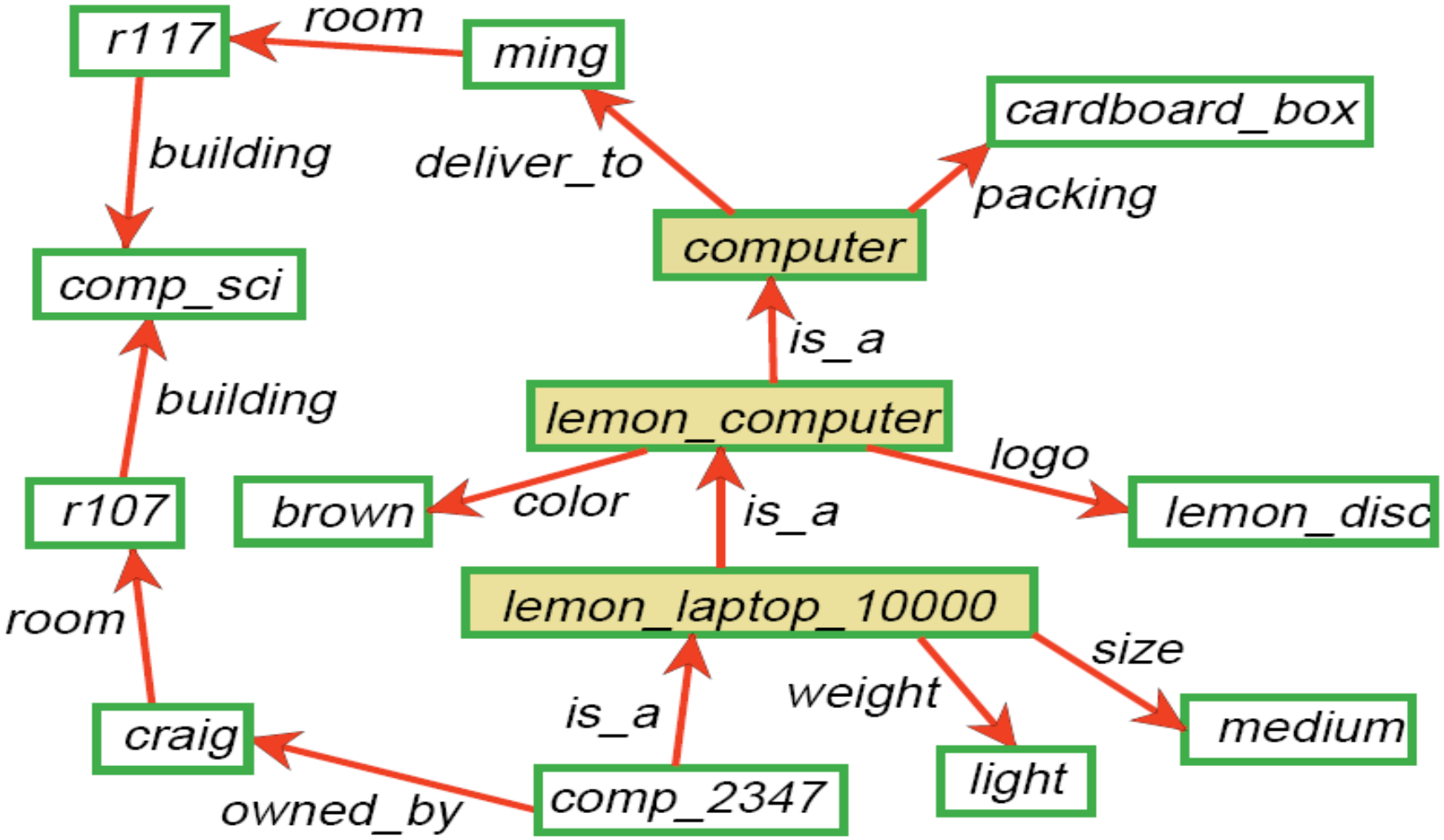
Relações primitivas e derivadas

- **Conhecimento primitivo**: definido explicitamente pelos fatos
- **Conhecimento derivado**: definido pelas regras
  - Permite uma representação mais compacta
  - Permite que conclusões sejam tiradas das observações do domínio
    - Importante porque não observamos tudo diretamente do domínio
    - Muito do que sabemos do domínio é inferido pelas observações e pelo conhecimento mais geral

Relações primitivas e derivadas

- Ex: Ao invés de especificar que o computador comp_2347 tem como logo um lemon disc, podemos saber que todos os computadores da marca Lemon tem este logo
- Associar as propriedades com as **classes** e não com os indivíduos
- Permitir um atributo especial **é_um** entre um indivíduo e uma classe ou entre duas classes
  - Permitindo **herança de propriedade**

Uma rede semântica estruturada

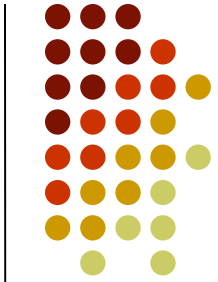


A lógica da herança de propriedade

- Uma arco *p* de uma classe *c* significa que todo indivíduo na classe tem valor *n* para o atributo *p*
  - *prop(Obj, p, n) <-- prop(Obj, é_um, c).*
  - *Exemplo:*

$$prop(X, weight, light) \leftarrow$$
$$prop(X, is\_a, lemon\_laptop\_10000).$$
$$prop(X, is\_a, lemon\_computer) \leftarrow$$
$$prop(X, is\_a, lemon\_laptop\_10000).$$

Herança múltipla



- Um indivíduo normalmente é membro de mais de uma classe
  - Ex: a mesma pessoa pode ser uma mãe, uma professora, uma árbitra de futebol...
- Um indivíduo pode herdar propriedades de todas as classes – **herança múltipla**
- Se existem valores *default* para as propriedades
  - Podemos ter um problema quando um indivíduo herda valores *default* conflitantes de classes diferentes: **problema da herança múltipla**

Escolhendo entre relações primitivas e derivadas

- Associar um valor de um atributo com a classe mais geral que contém o valor
- Não associe propriedades contingentes de uma classe com a classe
  - Ex: Pode ser que todos os computadores de uma empresa sejam de uma determinada marca, com gabinete preto.
    - Não é uma boa idéia associar a cor preta a todos os computadores
    - Podem ser comprados computadores de outras marcas com cores diferentes
- Axiomatize na direção **causal**. Queremos conhecimento que seja estável com as mudanças do mundo.
  - a ^ b, a ← b ou b ← a?

Escolhendo um procedimento de inferência

- O que afeta a eficiência da computação de uma solução para um problema?
  - Muitos algoritmos em IA são baseados em busca
    - Tamanho do espaço de busca
    - Fator de ramificação
    - Profundidade das soluções
  - Existência de boas heurísticas
    - Mínimos locais também são globais
    - Não existem platôs

Escolhendo um procedimento de inferência

- O que afeta a eficiência da computação de uma solução para um problema?
  - O algoritmo usado
    - Ele pode levar em consideração a informação disponibilizada pela heurística?
  - O nível de detalhes afeta diretamente a eficiência, pois modifica o espaço de busca
  - Talvez o fator que menos importa é a escolha da linguagem de programação
    - Provavelmente é melhor utilizar uma linguagem que foi projetada para o tipo de problema que queremos resolver