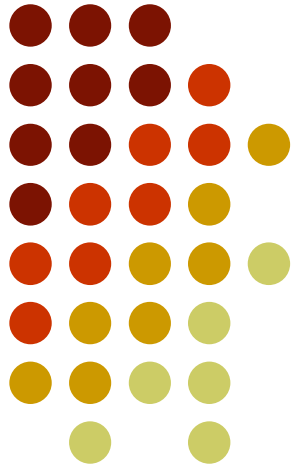
Representação de Conhecimento

Profa. Josiane M. P. Ferreira

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

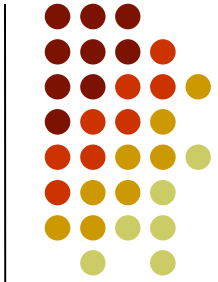
agosto/2008



Problemas e soluções

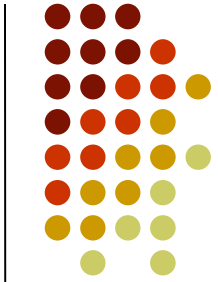
- Tipicamente o problema a ser resolvido ou a tarefa a ser feita, assim como o que constitui uma solução é dado informalmente:
  - Entregue os pacotes prontamente quando eles chegarem
  - Determine o que está errado com o sistema elétrico da casa
- Para resolver um problema precisamos:
  - Detalhar a tarefa e determinar o que constitui uma solução
  - Representar o problema em uma ling. que o computador entenda
  - E, então, usar um computador para resolvê-lo

O que considerar para resolver um problema



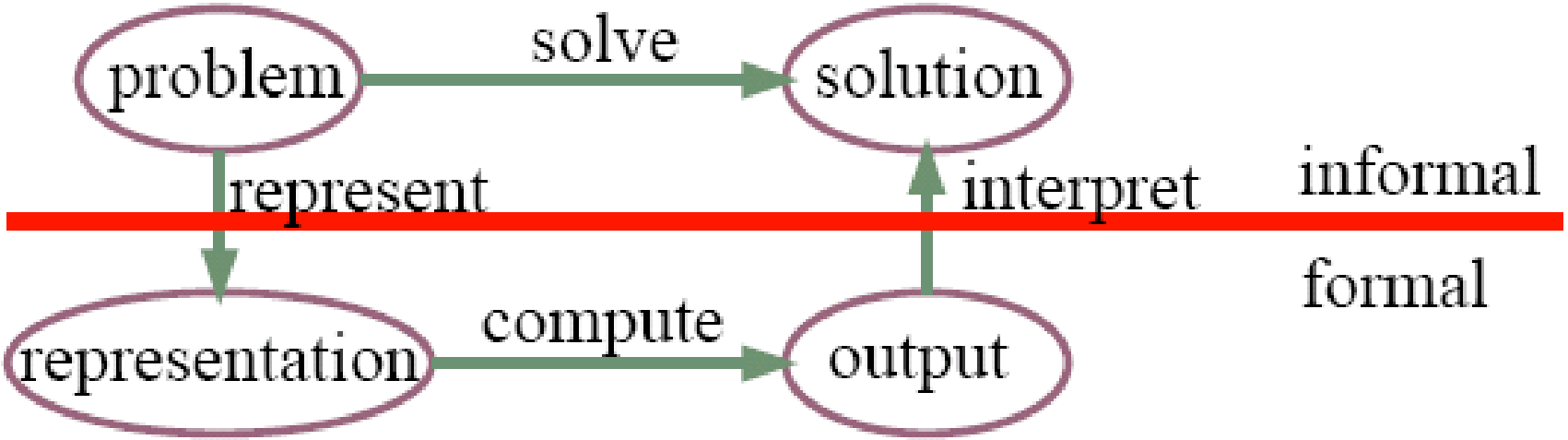
- O que é uma solução para o problema?
- O que é necessário na linguagem para representar o problema?
- Como podemos mapear a descrição informal do problema para uma representação?
- Quais distinções do mundo são importantes para resolver o problema?
- Qual é o conhecimento necessário?
- Qual é o nível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

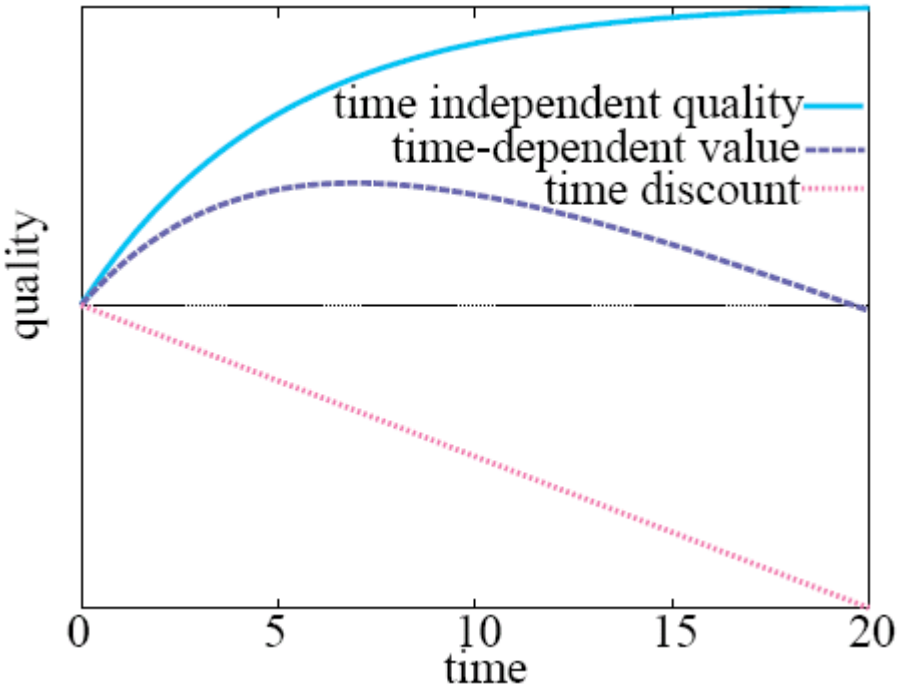
- Uma boa decisão é aquela que geralmente conduz a conseqüências boas.
  - Boas decisões podem ter conseqüências ruins. Decisões ruins podem ter conseqüências boas.
  - Exemplo: Um agente necessita de uma informação e decide acessar uma fonte de conhecimento
    - A informação está lá – conseqüência boa
    - A informação não está lá – conseqüência ruim
  - Uma **decisão boa** é uma ecolha que parece ser boa baseada na informação que o agente tem disponível.
  - Um **conseqüência boa** é o resultado de uma escolha que termina bem.

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

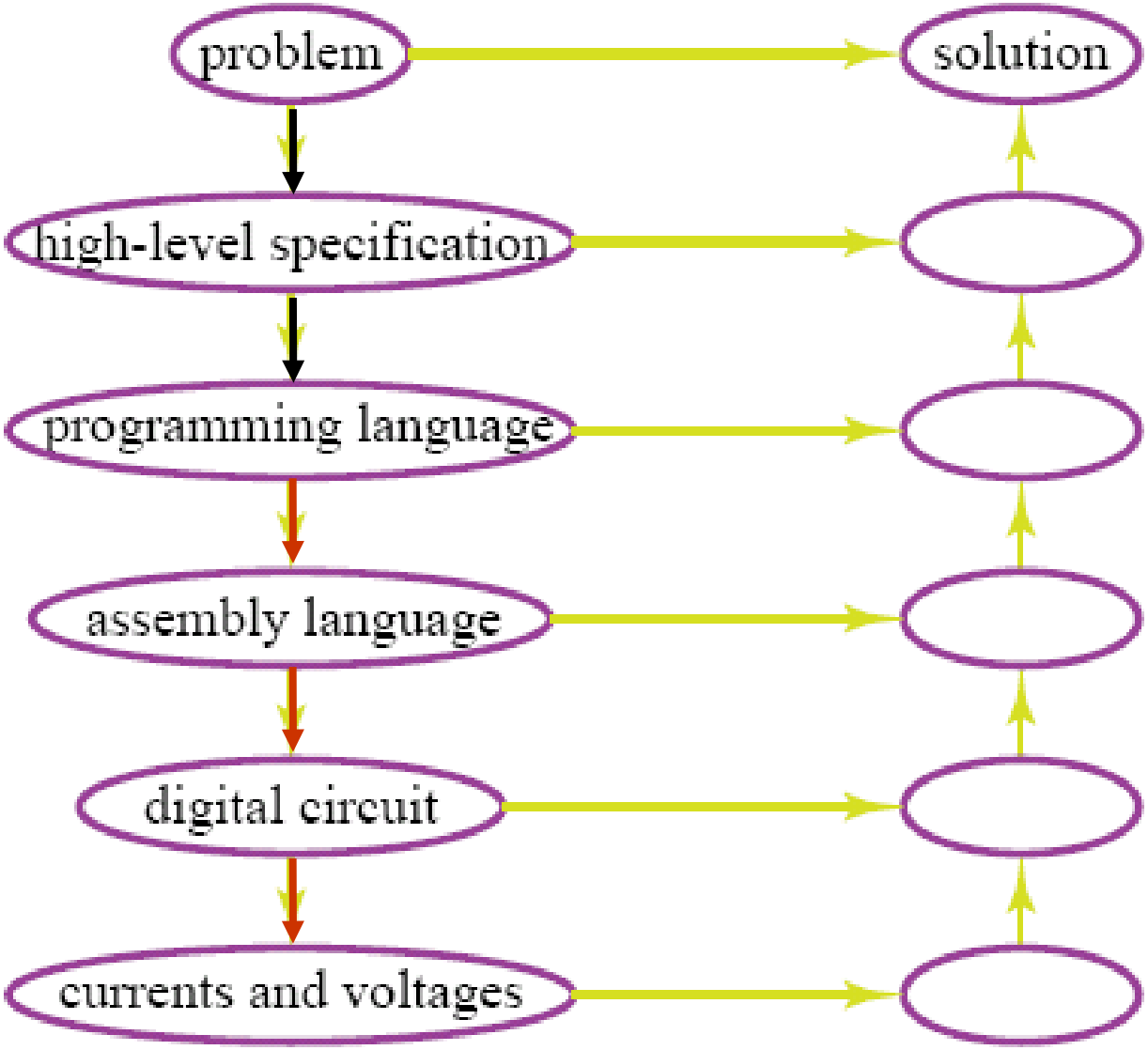
- Uma boa solução pode depender do tempo de inferência
  - Ex: A decisão de atravessar a rua deve ser rápida
- **Algoritmo a qualquer tempo**: um algoritmo para o qual a qualidade da solução melhora com o tempo
- Compromisso entre encontrar a melhor solução e encontrar uma resposta rapidamente



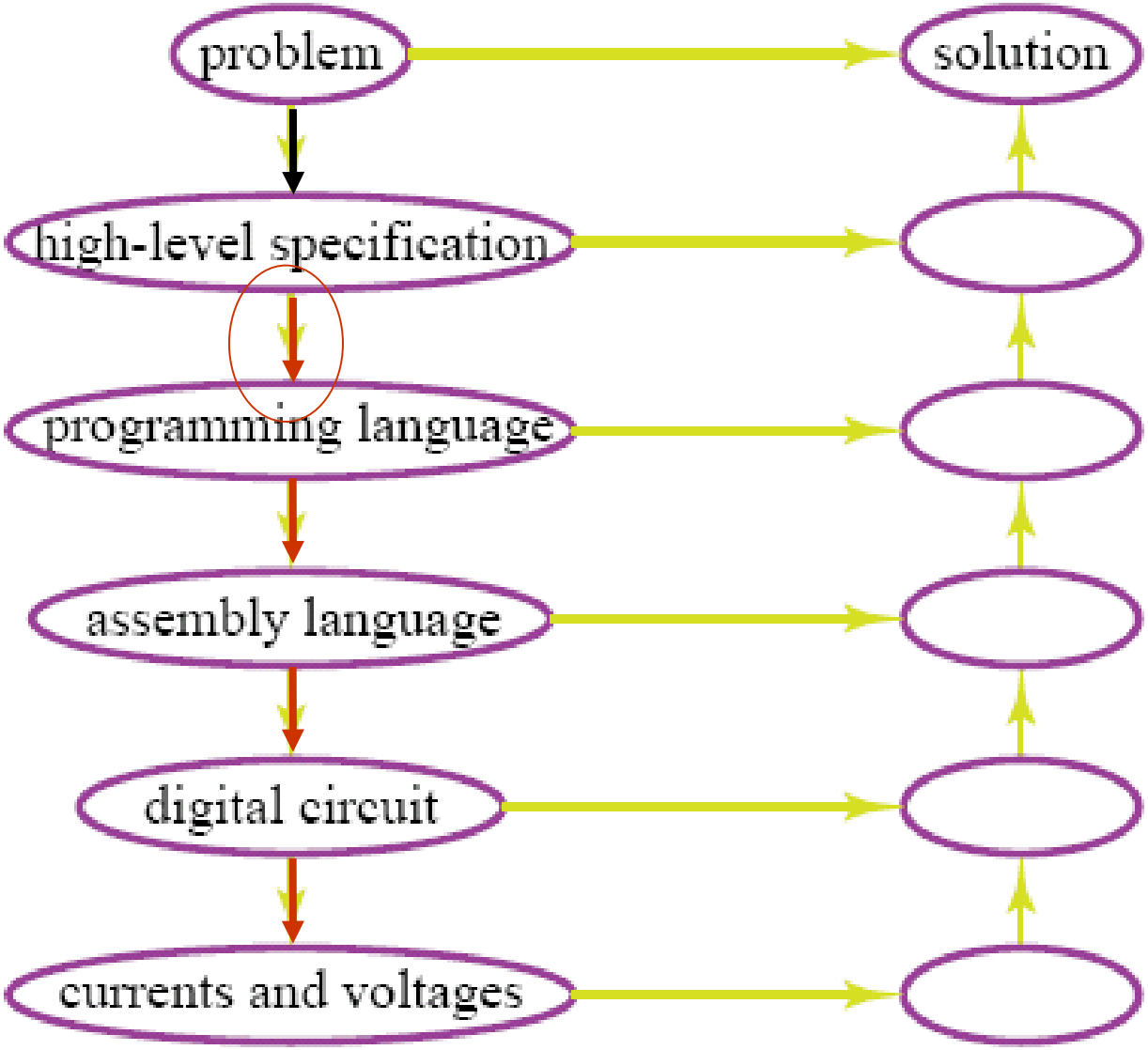
Escolhendo uma linguagem de representação

- 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

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 humano especificar e entender
- Uma descrição de baixo-nível pode ser mais precisa
  - 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
Representando com uma Linguagem Lógica

- Ex: vamos supor que "*red*" é uma categoria apropriada para classificar objetos
  - Podemos tratar "*red*" como uma relação unária e escrever que uma caneta é vermelha: ***red*(*pen*).**
    - Podemos perguntar o que é vermelho?: ?*red*(*X*).
    - Mas não podemos perguntar qual é a cor da caneta?: *?X(pen).*
  - Podemos considerar vermelho como uma constante e usar cor como um predicado: ***color(Obj, Val)***
    - "A caneta é vermelha": ***color(pen, red).***
    - Podemos perguntar qual é a cor da caneta?: *cor(pen, C).*
    - Mas não podemos perguntar qual propriedade da caneta tem valor *red*?": *?X(pen, red)*

Escolhendo objetos e relações
A relação objeto-atributo-valor

- A representação objeto-atributo-valor ***prop(Obj, Att, Val)*** significa que o objeto *Obj* tem o valor *Val* para o atributo *Att*
  - ***prop(pen, color, red)*** -> conseguimos responder todas as questões
    - O que é vermelho?: *?prop(X, color, red).*
    - Qual é a cor da caneta?: *?prop(pen, color, X).*
    - Qual propriedade da caneta tem valor *red*?": *?prop(pen, X, red).*
  - Para representar "*a* é um pacote"
    - *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
Representando relações mais complexas

- Como representar a relação *programado*(*C, A, H, S*), onde A é uma aula, C é um curso, H é a hora e S é a sala?
  - "aula 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, vamos **reificar** a atividade programada
    - **Reificar**: transformar em um objeto ("tornar real")
    - Precisamos inventar um índivíduo que tem um número de propriedades como curso, seção, hora e sala
      - Vamos chamar de **reserva** e nomear como uma *constante*, ***r123***:
        - *prop(r123, curso, cs422).*
        - *prop(r123, seção, 2).*
        - *prop(r123, hora, 1030).*
        - *prop(r123, sala, cc208).*
    - Uma representação modular e fácil de adicionar um novo atributo