(命题) 模态逻辑与模型检测

哲学数学计算机中的逻辑课程 (2016 年秋)

王彦晶

北大哲学系

2016 年 12 月 1 日 (国际艾滋病日)

处理知识变化

时态逻辑和模型检测

处理知识变化

回顾: 知识逻辑 (EPISTEMIC LOGIC)

推理知识 (与信念) 的模态逻辑 [von Wright 1951, Hintikka 1962].

- 语言: “主体 i 知道 (*knows that*) φ” ($\mathcal{K}_i\varphi$).
- 模型里对每个 i 有个关系 R_i.

回顾: 知识逻辑 (EPISTEMIC LOGIC)

推理知识 (与信念) 的模态逻辑 [von Wright 1951, Hintikka 1962].

- 语言: “主体 i 知道 (*knows that*) φ” ($\mathcal{K}_i\varphi$).
- 模型里对每个 i 有个关系 R_i.
- 关系要满足一些性质, 例如最强的要求: 自反传递对称 (等价关系, 直观上也可理解为 “不可区分关系”: 分不清 w 和 v).
- 在世界 w 上 i 知道 φ 当且仅当在所有从 w 出发 i 不可区分的世界上 φ 都为真.

例如, 假设纽约确实在下雨 (p),

回顾: 知识逻辑 (EPISTEMIC LOGIC)

推理知识 (与信念) 的模态逻辑 [von Wright 1951, Hintikka 1962].

- 语言: “主体 i 知道 (*knows that*) φ” ($\mathcal{K}_i\varphi$).
- 模型里对每个 i 有个关系 R_i.
- 关系要满足一些性质, 例如最强的要求: 自反传递对称 (等价关系, 直观上也可理解为 “不可区分关系”: 分不清 w 和 v).
- 在世界 w 上 i 知道 φ 当且仅当在所有从 w 出发 i 不可区分的世界上 φ 都为真.

例如, 假设纽约确实在下雨 (p), 但 1 不知道,

回顾: 知识逻辑 (EPISTEMIC LOGIC)

推理知识 (与信念) 的模态逻辑 [von Wright 1951, Hintikka 1962].

- 语言: "主体 i 知道 (*knows that*) φ" ($\mathcal{K}_i\varphi$).
- 模型里对每个 i 有个关系 R_i.
- 关系要满足一些性质, 例如最强的要求: 自反传递对称 (等价关系, 直观上也可理解为 "不可区分关系": 分不清 w 和 v).
- 在世界 w 上 i 知道 φ 当且仅当在所有从 w 出发 i 不可区分的世界上 φ 都为真.

例如, 假设纽约确实在下雨 (p), 但 1 不知道, 不过 1 知道 2 知道纽约是否在下雨...

$w \vDash p \wedge$

回顾: 知识逻辑 (EPISTEMIC LOGIC)

推理知识 (与信念) 的模态逻辑 [von Wright 1951, Hintikka 1962].

- 语言: "主体 i 知道 (*knows that*) φ" ($\mathcal{K}_i\varphi$).
- 模型里对每个 i 有个关系 R_i.
- 关系要满足一些性质, 例如最强的要求: 自反传递对称 (等价关系, 直观上也可理解为 "不可区分关系": 分不清 w 和 v).
- 在世界 w 上 i 知道 φ 当且仅当在所有从 w 出发 i 不可区分的世界上 φ 都为真.

例如, 假设纽约确实在下雨 (p), 但 1 不知道, 不过 1 知道 2 知道纽约是否在下雨...

$w \vDash p \wedge \neg\mathcal{K}_1 p \wedge \mathcal{K}_2 p \wedge \mathcal{K}_1(\mathcal{K}_2 p \vee \mathcal{K}_2\neg p) \wedge \mathcal{K}_2\neg\mathcal{K}_1 p \wedge \mathcal{K}_1\mathcal{K}_2\neg\mathcal{K}_1 p.$

逻辑工具主要用来处理人脑想不清楚的事情.

最强的 S5 系统 (对自反传递对称的框架类完全)

公理模式		推理规则	
TAUT	命题重言式	MP	$\dfrac{\varphi, \varphi \to \psi}{\psi}$
K	$\mathcal{K}_i(\varphi \to \psi) \to (\mathcal{K}_i\varphi \to \mathcal{K}_i\psi)$	NECK	$\dfrac{\varphi}{\mathcal{K}_i\varphi}$
T	$\mathcal{K}_i\varphi \to \varphi$		
4	$\mathcal{K}_i\varphi \to \mathcal{K}_i\mathcal{K}_i\varphi$		
5	$\neg\mathcal{K}_i\varphi \to \mathcal{K}_i\neg\mathcal{K}_i\varphi$		

如何理解这几条公理:

- T: 理想的知识应该是真的.
- 4, 5: 正负自省公理 (Introspection axioms).

知之为知之，不知为不知，是知也

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了!

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!
- 若没人站出来, 他就重复: 现在知道自己头上有泥巴的站出来!

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!
- 若没人站出来, 他就重复: 现在知道自己头上有泥巴的站出来!
- 当爸爸一共重复 k 次 (包括第一次) 之后, k 个头上有泥巴的小孩都突然站出来了.

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!
- 若没人站出来, 他就重复: 现在知道自己头上有泥巴的站出来!
- 当爸爸一共重复 k 次 (包括第一次) 之后, k 个头上有泥巴的小孩都突然站出来了. 为什么? (假设虎父无犬子)

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!
- 若没人站出来, 他就重复: 现在知道自己头上有泥巴的站出来!
- 当爸爸一共重复 k 次 (包括第一次) 之后, k 个头上有泥巴的小孩都突然站出来了. 为什么? (假设虎父无犬子)

- $k = n = 1$ 的时候, 显然.

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!
- 若没人站出来, 他就重复: 现在知道自己头上有泥巴的站出来!
- 当爸爸一共重复 k 次 (包括第一次) 之后, k 个头上有泥巴的小孩都突然站出来了. 为什么? (假设虎父无犬子)

- $k = n = 1$ 的时候, 显然.
- $k = 1, n = 2$ 的时候, 也比较显然.

泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!
- 若没人站出来, 他就重复: 现在知道自己头上有泥巴的站出来!
- 当爸爸一共重复 k 次 (包括第一次) 之后, k 个头上有泥巴的小孩都突然站出来了. 为什么? (假设虎父无犬子)

- $k = n = 1$ 的时候, 显然.
- $k = 1, n = 2$ 的时候, 也比较显然.
- $k = n = 2$ 的时候, 老爸的第一句宣告 “你们中间有人把泥巴弄到头上了!” 是废话么?

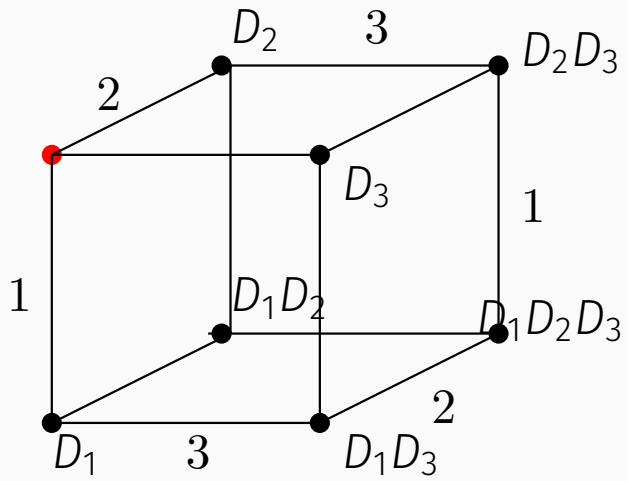
泥孩谜题 (MUDDY CHILDREN)

- n 个小孩在门口玩, 其中 $k \geq 1$ 额头弄上了泥巴.
- 他们只能看到别人头上有没有泥巴, 但是因为没有镜子, 看不到自己头上的情况.
- 他们的老爸 (逻辑学家) 出门看到孩子们玩的脏兮兮的很生气, 他说: 你们中间有人把泥巴弄到头上了! 接着他说: 知道自己头上有泥巴的给我站出来!
- 若没人站出来, 他就重复: 现在知道自己头上有泥巴的站出来!
- 当爸爸一共重复 k 次 (包括第一次) 之后, k 个头上有泥巴的小孩都突然站出来了. 为什么? (假设虎父无犬子)

- $k = n = 1$ 的时候, 显然.
- $k = 1, n = 2$ 的时候, 也比较显然.
- $k = n = 2$ 的时候, 老爸的第一句宣告 “你们中间有人把泥巴弄到头上了!” 是废话么? 不那么显然了吧?

当有三个脏小孩的情况

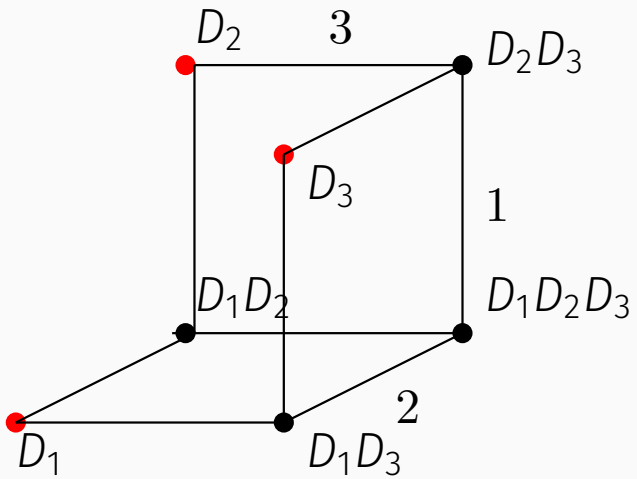
当前模型:



“至少有一个人头上有泥巴!” $\psi = D_1 \vee D_2 \vee D_3$

当有三个脏小孩的情况

当前模型:



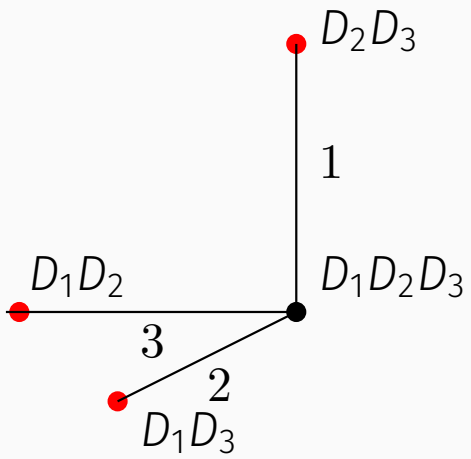
知道自己头上有泥巴的给我站出来!

没人 (敢) 往前迈步. 其实这就在说: 我们都不知道!

$\chi = \neg\mathcal{K}_1 D_1 \wedge \neg\mathcal{K}_2 D_2 \wedge \neg\mathcal{K}_3 D_3$

当有三个脏小孩的情况

当前模型:



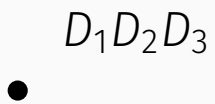
现在知道自己头上有泥巴的给我站出来!

还是没人站出来.

$\chi = \neg\mathcal{K}_1 D_1 \wedge \neg\mathcal{K}_2 D_2 \wedge \neg\mathcal{K}_3 D_3$

当有三个脏小孩的情况

当前模型:



现在所有小孩都知道他们脑袋上有泥巴了:

$D_1D_2D_3 \vDash K_1D_1 \wedge K_2D_2 \wedge K_3D_3$, 而且这也是公共知识.

重复宣告无知反而得到了知识!

公开宣告逻辑 (PUBLIC ANNOUNCEMENT LOGIC) BY PLAZA (1989)

公开宣告逻辑的语言 ($p \in \mathsf{PV}, i \in \mathsf{I}$):

$$\varphi \quad ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{K}_i\varphi \mid [!\varphi]\varphi$$

公开宣告逻辑 (PUBLIC ANNOUNCEMENT LOGIC) BY PLAZA (1989)

公开宣告逻辑的语言 ($p \in \mathsf{PV}, i \in \mathsf{I}$):

$$\varphi \quad ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{K}_i\varphi \mid [!\varphi]\varphi$$

语义也是通过 (S5) 克里普克模型给出 $\mathcal{M} = (W, \{R_i\}_{i\in I}, V)$:

$$\mathcal{M}, w \vDash [!\psi]\varphi \quad \Leftrightarrow \quad \text{若} \mathcal{M}, w \vDash \psi \text{ 则} \mathcal{M}|_\psi, w \vDash \varphi$$

公开宣告逻辑 (PUBLIC ANNOUNCEMENT LOGIC) BY PLAZA (1989)

公开宣告逻辑的语言 ($p \in \mathsf{PV}, i \in \mathsf{I}$):

$$\varphi \quad ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{K}_i\varphi \mid [!\varphi]\varphi$$

语义也是通过 (S5) 克里普克模型给出 $\mathcal{M} = (W, \{R_i\}_{i\in I}, V)$:

$$\boxed{\mathcal{M}, w \vDash [!\psi]\varphi \quad \Leftrightarrow \quad \text{若} \mathcal{M}, w \vDash \psi \text{ 则} \mathcal{M}|_\psi, w \vDash \varphi}$$

其中 $\mathcal{M}|_\psi = (W', \{R'_i \mid i \in \mathsf{I}\}, V')$ 使得: $W' = \{w \mid \mathcal{M}, w \vDash \psi\}$, $R'_i = R_i|_{W'\times W'}$ 且 $V'(p) = V(p) \cap W'$.

公开宣告逻辑 (PUBLIC ANNOUNCEMENT LOGIC) BY PLAZA (1989)

公开宣告逻辑的语言 ($p \in \mathsf{PV}, i \in \mathsf{I}$):

$$\varphi \quad ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{K}_i\varphi \mid [!\varphi]\varphi$$

语义也是通过 (S5) 克里普克模型给出 $\mathcal{M} = (W, \{R_i\}_{i\in I}, V)$:

$$\boxed{\mathcal{M}, w \vDash [!\psi]\varphi \quad \Leftrightarrow \quad \text{若} \mathcal{M}, w \vDash \psi \text{ 则} \mathcal{M}|_\psi, w \vDash \varphi}$$

其中 $\mathcal{M}|_\psi = (W', \{R'_i \mid i \in \mathsf{I}\}, V')$ 使得: $W' = \{w \mid \mathcal{M}, w \vDash \psi\}$, $R'_i = R_i|_{W'\times W'}$ 且 $V'(p) = V(p) \cap W'$.其实就是做了一个子模型.

公开宣告逻辑 (PUBLIC ANNOUNCEMENT LOGIC) BY PLAZA (1989)

公开宣告逻辑的语言 ($p \in \mathsf{PV}, i \in \mathsf{I}$):

$$\varphi \quad ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{K}_i\varphi \mid [!\varphi]\varphi$$

语义也是通过 (S5) 克里普克模型给出 $\mathcal{M} = (W, \{R_i\}_{i\in I}, V)$:

$$\boxed{\mathcal{M}, w \vDash [!\psi]\varphi \quad \Leftrightarrow \quad \text{若} \mathcal{M}, w \vDash \psi \text{ 则} \mathcal{M}|_\psi, w \vDash \varphi}$$

其中 $\mathcal{M}|_\psi = (W', \{R'_i \mid i \in \mathsf{I}\}, V')$ 使得: $W' = \{w \mid \mathcal{M}, w \vDash \psi\}$, $R'_i = R_i|_{W'\times W'}$ 且 $V'(p) = V(p) \cap W'$.其实就是做了一个子模型.

动作的意义在于改变!

公开宣告逻辑 (PUBLIC ANNOUNCEMENT LOGIC) BY PLAZA (1989)

公开宣告逻辑的语言 ($p \in \mathsf{PV}, i \in \mathsf{I}$):

$$\varphi \quad ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{K}_i\varphi \mid [!\varphi]\varphi$$

语义也是通过 (S5) 克里普克模型给出 $\mathcal{M} = (W, \{R_i\}_{i\in I}, V)$:

$$\boxed{\mathcal{M}, w \vDash [!\psi]\varphi \quad \Leftrightarrow \quad \text{若} \mathcal{M}, w \vDash \psi \text{ 则} \mathcal{M}|_\psi, w \vDash \varphi}$$

其中 $\mathcal{M}|_\psi = (W', \{R_i' \mid i \in \mathsf{I}\}, V')$ 使得: $W' = \{w \mid \mathcal{M}, w \vDash \psi\}$, $R_i' = R_i|_{W'\times W'}$ 且 $V'(p) = V(p) \cap W'$.其实就是做了一个子模型.

动作的意义在于改变! 如下模型有: $\mathcal{M}, w_1 \vDash \neg\mathcal{K}_1 p \wedge [!p]\mathcal{K}_1 p$.

公开宣告逻辑 (PUBLIC ANNOUNCEMENT LOGIC) BY PLAZA (1989)

公开宣告逻辑的语言 ($p \in \mathsf{PV}, i \in \mathsf{I}$):

$$\varphi \quad ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{K}_i\varphi \mid [!\varphi]\varphi$$

语义也是通过 (S5) 克里普克模型给出 $\mathcal{M} = (W, \{R_i\}_{i\in I}, V)$:

$$\boxed{\mathcal{M}, w \vDash [!\psi]\varphi \quad \Leftrightarrow \quad \text{若} \mathcal{M}, w \vDash \psi \text{ 则} \mathcal{M}|_\psi, w \vDash \varphi}$$

其中 $\mathcal{M}|_\psi = (W', \{R_i' \mid i \in \mathsf{I}\}, V')$ 使得: $W' = \{w \mid \mathcal{M}, w \vDash \psi\}$, $R_i' = R_i|_{W'\times W'}$ 且 $V'(p) = V(p) \cap W'$.其实就是做了一个子模型.

动作的意义在于改变! 如下模型有: $\mathcal{M}, w_1 \vDash \neg\mathcal{K}_1 p \wedge [!p]\mathcal{K}_1 p$.

刚才 3 个泥孩的谜题可形式化为 ($\mathcal{M}$ 是初始模型):

$\mathcal{M}, (D_1D_2D_3) \vDash [!\psi][!\chi][!\chi](\mathcal{K}_1D_1 \wedge \mathcal{K}_2D_2 \wedge \mathcal{K}_3D_3)$

不是所有被宣告的都会变成公共知识

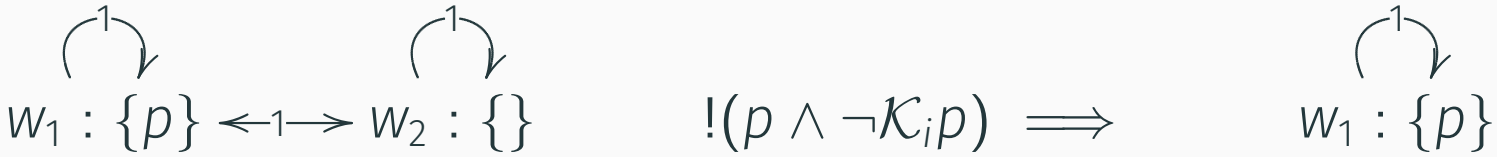
$[\varphi]\mathcal{K}_i\varphi$ 不是有效的!

不是所有被宣告的都会变成公共知识

$[\varphi]\mathcal{K}_i\varphi$ 不是有效的! 事实上连 $[\varphi]\varphi$ 都不是有效的!

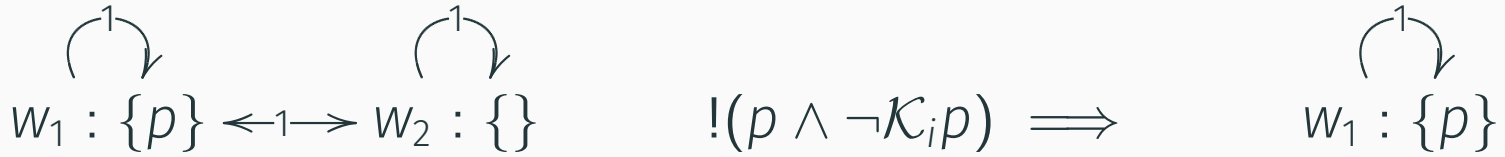
不是所有被宣告的都会变成公共知识

$[\varphi]\mathcal{K}_i\varphi$ 不是有效的! 事实上连 $[\varphi]\varphi$ 都不是有效的! 有的真的公式被宣告之后会变成假的!



不是所有被宣告的都会变成公共知识

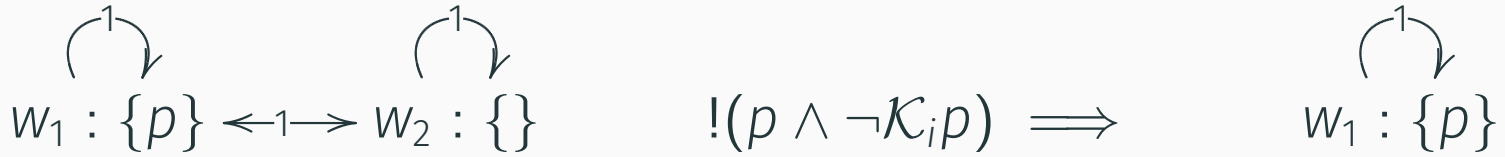
$[\varphi]\mathcal{K}_i\varphi$ 不是有效的! 事实上连 $[\varphi]\varphi$ 都不是有效的! 有的真的公式被宣告之后会变成假的!



$w_1 \vDash (p \wedge \neg\mathcal{K}_1 p) \wedge [p \wedge \neg\mathcal{K}_1 p]\mathcal{K}_1 p.$

不是所有被宣告的都会变成公共知识

$[\varphi]\mathcal{K}_i\varphi$ 不是有效的! 事实上连 $[\varphi]\varphi$ 都不是有效的! 有的真的公式被宣告之后会变成假的!

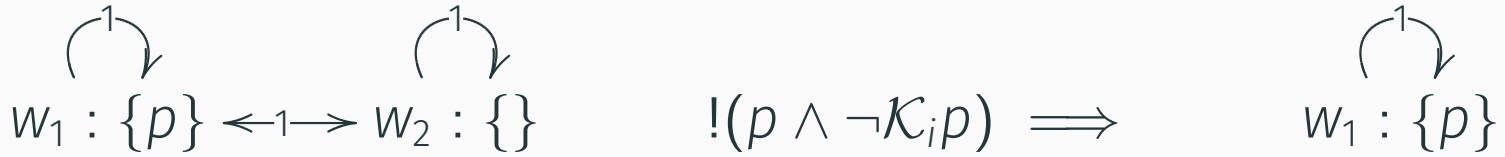


$w_1 \vDash (p \wedge \neg\mathcal{K}_1 p) \wedge [p \wedge \neg\mathcal{K}_1 p]\mathcal{K}_1 p.$

哪些公式是在公开宣告下保持的?

不是所有被宣告的都会变成公共知识

$[\varphi]\mathcal{K}_i\varphi$ 不是有效的! 事实上连 $[\varphi]\varphi$ 都不是有效的! 有的真的公式被宣告之后会变成假的!



$w_1 \vDash (p \wedge \neg\mathcal{K}_1 p) \wedge [p \wedge \neg\mathcal{K}_1 p]\mathcal{K}_1 p$.

哪些公式是在公开宣告下保持的?

有时候假话说出来之后会使得它自己变成真的 (“True lies”).

公开宣告逻辑的公理化

Plaza (1989) 提出了一个公理化, 将 PAL 规约到知识逻辑中.

公开宣告逻辑的公理化

Plaza (1989) 提出了一个公理化, 将 PAL 规约到知识逻辑中. 很有趣, 但不够直观, 证明方法也不够一般.

PAN 系统 (Wang & Cao 2013) 省略了 T,4,5

Axiom Schemas

TAUT	命题重言式
DISTK	$\mathcal{K}(\varphi \to \chi) \to (\mathcal{K}\varphi \to \mathcal{K}\chi)$
DISTA	$[!\psi](\varphi \to \chi) \to ([!\psi]\varphi \to [!\psi]\chi)$
INV	$(p \to [!\psi]p) \wedge (\neg p \to [!\psi]\neg p)$
EXE	$\langle !\psi\rangle \top \leftrightarrow \psi$
PR	$\mathcal{K}[!\psi]\varphi \to [!\psi]\mathcal{K}\varphi$
NL	$\langle !\psi\rangle \mathcal{K}\varphi \to \mathcal{K}[!\psi]\varphi$

Rules

MP	$\dfrac{\varphi, \varphi \to \psi}{\psi}$
NECK	$\dfrac{\varphi}{\mathcal{K}\varphi}$
NECA	$\dfrac{\varphi}{[!\psi]\varphi}$

这种的谜题都可以类似的形式化并自动解决

- Cheryl's Birthday
- Unfaithful wives/husbands
- Sum and Product
- Russian cards
- Puzzle of three hats
- 100 prisoner and a light bulb
- The hardest logic puzzle ever
- ...

CHERYL'S BIRTHDAY

艾伯特和柏纳刚认识谢丽尔，想要知道谢丽尔的生日，谢丽尔列出了十个可能的日期：

5 月 15 日、5 月 16 日、5 月 19 日、6 月 17 日、6 月 18 日、7 月 14 日、7 月 16 日、8 月 14 日、8 月 15 日、8 月 17 日

接着谢丽尔分别告诉艾伯特及柏纳她生日的月及日，以下是艾伯特和柏纳的对话。

艾伯特：我不知道谢丽尔的生日是哪一天，但我知道你也不知道

柏纳：一开始我不知道谢丽尔的生日，但现在我知道了

艾伯特：那我也知道谢丽尔的生日了

请问谢丽尔的生日是那一天?

RUSSIAN CARDS

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌.

RUSSIAN CARDS

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌. 问题: B 和 C 有没有可能通过公开的 (真实的) 宣告使得互相都知道对方的牌是什么, 但同时 E 还是不知道任何一张不在手里的牌的归属?

RUSSIAN CARDS

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌. 问题: B 和 C 有没有可能通过公开的 (真实的) 宣告使得互相都知道对方的牌是什么, 但同时 E 还是不知道任何一张不在手里的牌的归属?

比如: 假设 B 拿的是 123, C 那的是 456, E 拿的是 7. B 说 “我或者 C 拿的是 123”, 然后 C 说 “我或者 B 拿的是 456”, 这样行么?

RUSSIAN CARDS

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌. 问题: B 和 C 有没有可能通过公开的 (真实的) 宣告使得互相都知道对方的牌是什么, 但同时 E 还是不知道任何一张不在手里的牌的归属?

比如: 假设 B 拿的是 123, C 那的是 456, E 拿的是 7. B 说 “我或者 C 拿的是 123”, 然后 C 说 “我或者 B 拿的是 456”, 这样行么?

不行!

RUSSIAN CARDS

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌. 问题: B 和 C 有没有可能通过公开的 (真实的) 宣告使得互相都知道对方的牌是什么, 但同时 E 还是不知道任何一张不在手里的牌的归属?

比如: 假设 B 拿的是 123, C 那的是 456, E 拿的是 7. B 说 “我或者 C 拿的是 123”, 然后 C 说 “我或者 B 拿的是 456”, 这样行么?

不行! 因为宣告者还得保证宣告是真的, 这也就意味着 B 知道: B 拿的是 123 或者 C 拿的是 123 ($[!\mathcal{K}_B(123_B \vee 123_C)]$) . 可是 B 怎么可能知道 C 拿的是什么牌? 所以 B 肯定手里拿的是 123 (有跳步).

RUSSIAN CARDS

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌. 问题: B 和 C 有没有可能通过公开的 (真实的) 宣告使得互相都知道对方的牌是什么, 但同时 E 还是不知道任何一张不在手里的牌的归属?

比如: 假设 B 拿的是 123, C 那的是 456, E 拿的是 7. B 说 “我或者 C 拿的是 123”, 然后 C 说 “我或者 B 拿的是 456”, 这样行么?

不行! 因为宣告者还得保证宣告是真的, 这也就意味着 B 知道: B 拿的是 123 或者 C 拿的是 123 ($[!\mathcal{K}_B(123_B \vee 123_C)]$) . 可是 B 怎么可能知道 C 拿的是什么牌? 所以 B 肯定手里拿的是 123 (有跳步).

同时, 我们还需要保证拿到任何三张牌都可以有办法让对家知道.

RUSSIAN CARDS

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌. 问题: B 和 C 有没有可能通过公开的 (真实的) 宣告使得互相都知道对方的牌是什么, 但同时 E 还是不知道任何一张不在手里的牌的归属?

比如: 假设 B 拿的是 123, C 那的是 456, E 拿的是 7. B 说 “我或者 C 拿的是 123”, 然后 C 说 “我或者 B 拿的是 456”, 这样行么?

不行! 因为宣告者还得保证宣告是真的, 这也就意味着 B 知道: B 拿的是 123 或者 C 拿的是 123 ($[!\mathcal{K}_B(123_B \vee 123_C)]$) . 可是 B 怎么可能知道 C 拿的是什么牌? 所以 B 肯定手里拿的是 123 (有跳步).

同时, 我们还需要保证拿到任何三张牌都可以有办法让对家知道. 甚至即使 E 知道我们的宣告方式, 他还是不能知道我们手里的牌!

从 1, 2, 3, 4, 5, 6, 7 中 B 和 C 分别被保密的发了三张牌, 而 E 拿走剩下的牌. 问题: B 和 C 有没有可能通过公开的 (真实的) 宣告使得互相都知道对方的牌是什么, 但同时 E 还是不知道任何一张不在手里的牌的归属?

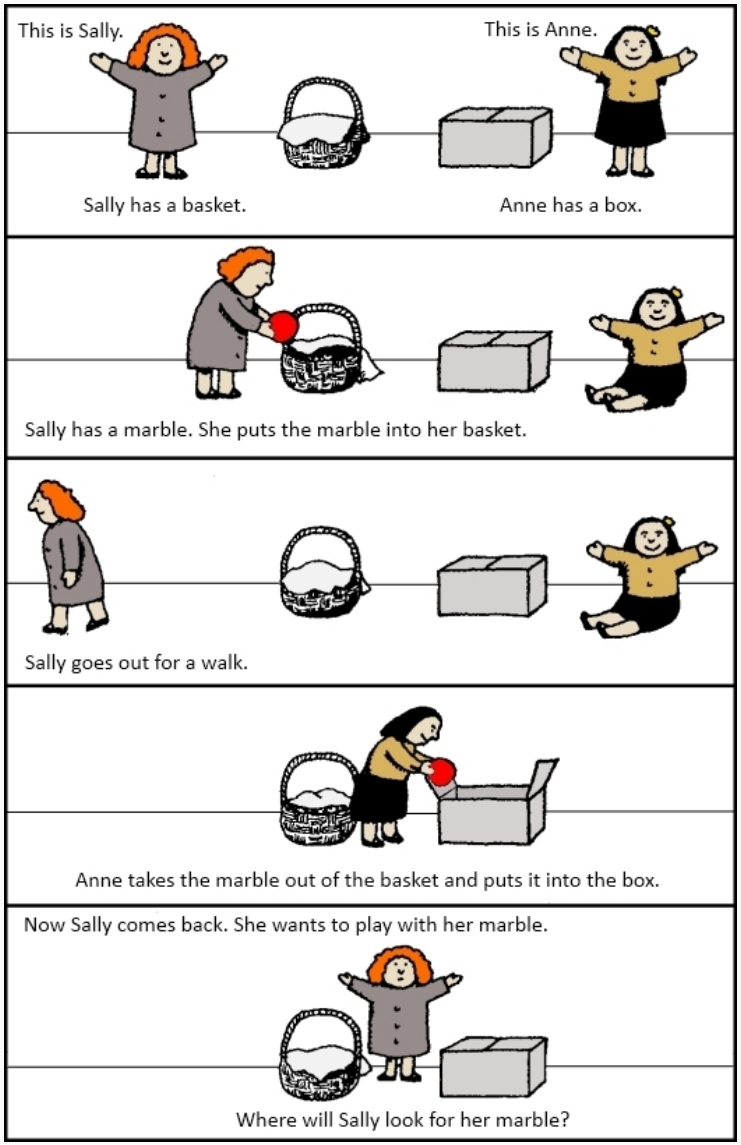
比如: 假设 B 拿的是 123, C 那的是 456, E 拿的是 7. B 说 “我或者 C 拿的是 123”, 然后 C 说 “我或者 B 拿的是 456”, 这样行么?

不行! 因为宣告者还得保证宣告是真的, 这也就意味着 B 知道: B 拿的是 123 或者 C 拿的是 123 ($[!\mathcal{K}_B(123_B \vee 123_C)]$) . 可是 B 怎么可能知道 C 拿的是什么牌? 所以 B 肯定手里拿的是 123 (有跳步).

同时, 我们还需要保证拿到任何三张牌都可以有办法让对家知道. 甚至即使 E 知道我们的宣告方式, 他还是不能知道我们手里的牌!

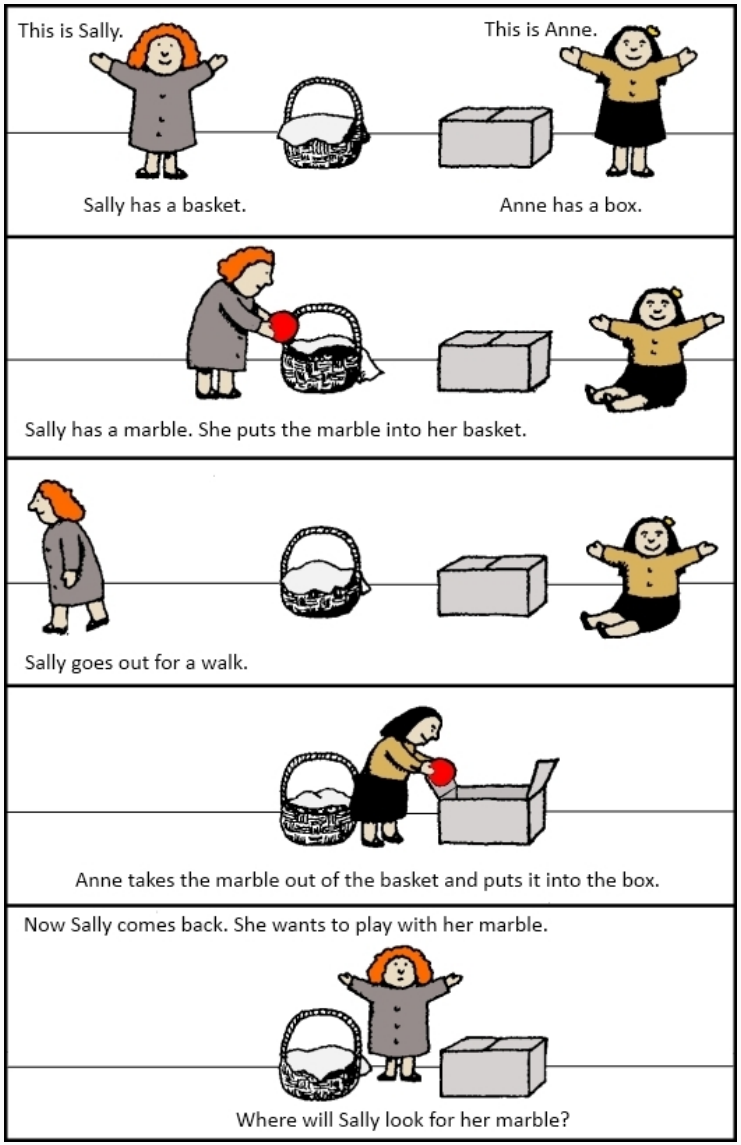
“纸牌密码学”(Card Cryptography)

THEORY OF MIND (心理理论) 理解别人的错误信念



Baron-Cohen, Leslie and Frith (1985)

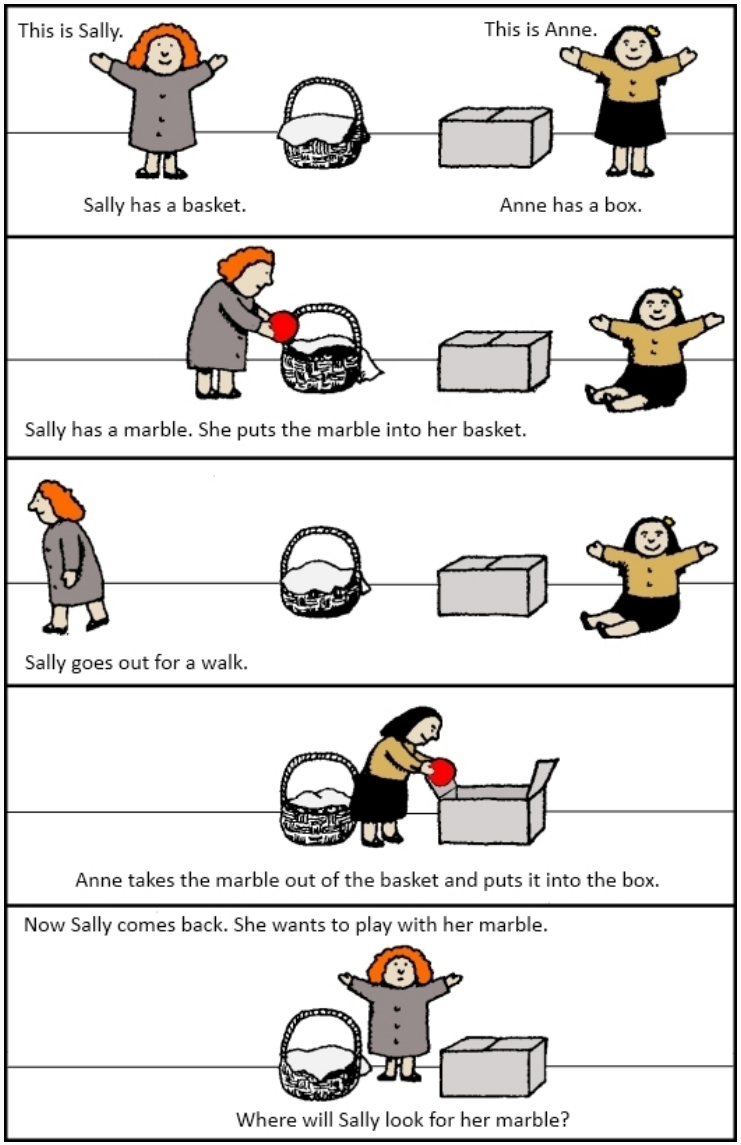
THEORY OF MIND (心理理论) 理解别人的错误信念



Baron-Cohen, Leslie and Frith (1985)

Dumontheil et al. (2010)

THEORY OF MIND (心理理论) 理解别人的错误信念



Baron-Cohen, Leslie and Frith (1985)

Dumontheil et al. (2010)

“把大球往上拿一格!”到底拿哪个要看是谁说的

超越 “知道如是”(KNOWING THAT) 的知识逻辑

- 我知道是否这个定理是对的 (knowing whether).
- 我知道这瓶牛奶的价格是什么 (knowing what).
- 我知道怎么去清华 (knowing how).
- 我知道为什么他来晚了 (knowing why).

注意: 上面的 “知道” 不能换成 “相信”,

超越 “知道如是”(KNOWING THAT) 的知识逻辑

- 我知道是否这个定理是对的 (knowing whether).
- 我知道这瓶牛奶的价格是什么 (knowing what).
- 我知道怎么去清华 (knowing how).
- 我知道为什么他来晚了 (knowing why).

注意: 上面的 “知道” 不能换成 “相信”, 如果知识至少是信念为什么不能换?

超越 “知道如是”(KNOWING THAT) 的知识逻辑

- 我知道是否这个定理是对的 (knowing whether).
- 我知道这瓶牛奶的价格是什么 (knowing what).
- 我知道怎么去清华 (knowing how).
- 我知道为什么他来晚了 (knowing why).

注意: 上面的 “知道” 不能换成 “相信”, 如果知识至少是信念为什么不能换?

引入新的模态词, 给语言学上说得过去的语义. 得到本质上是一阶或二阶模态逻辑可判定片段的逻辑, 有很多有趣的技术及哲学问题.

超越 “知道如是”(KNOWING THAT) 的知识逻辑

- 我知道是否这个定理是对的 (knowing whether).
- 我知道这瓶牛奶的价格是什么 (knowing what).
- 我知道怎么去清华 (knowing how).
- 我知道为什么他来晚了 (knowing why).

注意: 上面的 “知道” 不能换成 “相信”, 如果知识至少是信念为什么不能换?

引入新的模态词, 给语言学上说得过去的语义. 得到本质上是一阶或二阶模态逻辑可判定片段的逻辑, 有很多有趣的技术及哲学问题.

详见本人主页及” 知识的逻辑” 课程.

知识与其他模态词的整合

- 知识与时间
- 知识与道义
- 知识与偏好
- 知识与策略

时态逻辑和模型检测

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$
- 分叉时间 (Branching time) 时态逻辑 (CTL): $EX\varphi$, $EG\varphi$, $\psi EU\varphi$

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$
- 分叉时间 (Branching time) 时态逻辑 (CTL): $EX\varphi$, $EG\varphi$, $\psi EU\varphi$
- 时态逻辑的好朋友: (Propositional) Dynamic Logic (PDL): $[\pi]\varphi$, 这里 π 是一个正则表达式, $\pi ::= a \mid (\pi;\pi) \mid (\pi + \pi) \mid \pi^*$

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$
- 分叉时间 (Branching time) 时态逻辑 (CTL): $EX\varphi$, $EG\varphi$, $\psi EU\varphi$
- 时态逻辑的好朋友: (Propositional) Dynamic Logic (PDL): $[\pi]\varphi$, 这里 π 是一个正则表达式, $\pi ::= a \mid (\pi;\pi) \mid (\pi + \pi) \mid \pi^*$

例如, 操作系统资源 (内存) 分配的管理程序需要处理进程 (process) i ($i = 1, 2$) 的请求 (req_i) 和分配 (own_i), 每个 (无穷) 执行路径要满足

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$
- 分叉时间 (Branching time) 时态逻辑 (CTL): $EX\varphi$, $EG\varphi$, $\psi EU\varphi$
- 时态逻辑的好朋友: (Propositional) Dynamic Logic (PDL): $[\pi]\varphi$, 这里 π 是一个正则表达式, $\pi ::= a \mid (\pi;\pi) \mid (\pi + \pi) \mid \pi^*$

例如, 操作系统资源 (内存) 分配的管理程序需要处理进程 (process) i ($i = 1, 2$) 的请求 (req_i) 和分配 (own_i), 每个 (无穷) 执行路径要满足

- $G\neg(own_1 \wedge own_2)$ 同一个资源不能同时分给两个进程

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$
- 分叉时间 (Branching time) 时态逻辑 (CTL): $EX\varphi$, $EG\varphi$, $\psi EU\varphi$
- 时态逻辑的好朋友: (Propositional) Dynamic Logic (PDL): $[\pi]\varphi$, 这里 π 是一个正则表达式, $\pi ::= a \mid (\pi;\pi) \mid (\pi + \pi) \mid \pi^*$

例如, 操作系统资源 (内存) 分配的管理程序需要处理进程 (process) i ($i = 1, 2$) 的请求 (req_i) 和分配 (own_i), 每个 (无穷) 执行路径要满足

- $G\neg(own_1 \wedge own_2)$ 同一个资源不能同时分给两个进程
- $G(req_i \rightarrow F\ own_i)$ 资源请求肯定会在未来某个时刻被满足

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$
- 分叉时间 (Branching time) 时态逻辑 (CTL): $EX\varphi$, $EG\varphi$, $\psi EU\varphi$
- 时态逻辑的好朋友: (Propositional) Dynamic Logic (PDL): $[\pi]\varphi$, 这里 π 是一个正则表达式, $\pi ::= a \mid (\pi;\pi) \mid (\pi + \pi) \mid \pi^*$

例如, 操作系统资源 (内存) 分配的管理程序需要处理进程 (process) i ($i = 1, 2$) 的请求 (req_i) 和分配 (own_i), 每个 (无穷) 执行路径要满足

- $G\neg(own_1 \wedge own_2)$ 同一个资源不能同时分给两个进程
- $G(req_i \rightarrow F\ own_i)$ 资源请求肯定会在未来某个时刻被满足
- $GF(req_1 \wedge \neg(own_1 \vee own_2)) \rightarrow GF\ own_1$ 如果进程 1 无穷多次的在资源闲置时申请, 则他会无穷多次的得到这个资源.

时态逻辑 (TEMPORAL LOGIC) 命题逻辑之上加上...

- 线性时间 (Linear time) 时态逻辑 (LTL): $X\varphi$ (Next), $\varphi U\psi$ (Until), $F\varphi := \top U\varphi$ (未来), $G\varphi := \neg F\neg\varphi$(永远), $\varphi W\psi := (\varphi U\psi) \vee G\varphi$
- 分叉时间 (Branching time) 时态逻辑 (CTL): $EX\varphi$, $EG\varphi$, $\psi EU\varphi$
- 时态逻辑的好朋友: (Propositional) Dynamic Logic (PDL): $[\pi]\varphi$, 这里 π 是一个正则表达式, $\pi ::= a \mid (\pi;\pi) \mid (\pi + \pi) \mid \pi^*$

例如, 操作系统资源 (内存) 分配的管理程序需要处理进程 (process) i ($i = 1, 2$) 的请求 (req_i) 和分配 (own_i), 每个 (无穷) 执行路径要满足

- $G\neg(own_1 \wedge own_2)$ 同一个资源不能同时分给两个进程
- $G(req_i \rightarrow F\ own_i)$ 资源请求肯定会在未来某个时刻被满足
- $GF(req_1 \wedge \neg(own_1 \vee own_2)) \rightarrow GF\ own_1$ 如果进程 1 无穷多次的在资源闲置时申请, 则他会无穷多次的得到这个资源.
- $G(req_1 \wedge req_2 \rightarrow (\neg own_2\ W(own_2\ W(\neg own_2\ W\ own_1))))$ 两个进程都申请, 则进程 2 最多在 1 得到资源前得到一次.

LTL 的语义

LTL 公式的语义定义在 (无穷) 序列 $\sigma = s_0 s_1 s_2 \ldots$ 上 (每个点上有基本命题赋值 V), 令 σ_k 为从 σ 的 k 位重新开始的 σ 的子串:

LTL 的语义

LTL 公式的语义定义在 (无穷) 序列 $\sigma = s_0 s_1 s_2 \ldots$ 上 (每个点上有基本命题赋值 V), 令 σ_k 为从 σ 的 k 位重新开始的 σ 的子串:

$$
\begin{array}{rcl}
\sigma \vDash p & \Leftrightarrow & p \in V(s_0) \\
\sigma \vDash X\varphi & \Leftrightarrow & \sigma_1 \vDash \varphi \\
\sigma \vDash \psi U \varphi & \Leftrightarrow & \underline{\text{存在}} \text{ 一个 } k \text{ 使得 } \sigma_k \vDash \varphi \text{ 并且对所有 } j < k, \sigma_j \vDash \psi
\end{array}
$$

LTL 的语义

LTL 公式的语义定义在 (无穷) 序列 $\sigma = s_0 s_1 s_2 \ldots$ 上 (每个点上有基本命题赋值 V), 令 σ_k 为从 σ 的 k 位重新开始的 σ 的子串:

$$
\begin{array}{rcl}
\sigma \vDash p & \Leftrightarrow & p \in V(s_0) \\
\sigma \vDash X\varphi & \Leftrightarrow & \sigma_1 \vDash \varphi \\
\sigma \vDash \psi U \varphi & \Leftrightarrow & \underline{\text{存在}} \text{ 一个 } k \text{ 使得 } \sigma_k \vDash \varphi \text{ 并且对所有 } j < k, \sigma_j \vDash \psi
\end{array}
$$

- 回忆: $\varphi W \psi := (\varphi U \psi) \vee G\varphi$

LTL 的语义

LTL 公式的语义定义在 (无穷) 序列 $\sigma = s_0 s_1 s_2 \ldots$ 上 (每个点上有基本命题赋值 V), 令 σ_k 为从 σ 的 k 位重新开始的 σ 的子串:

$$
\begin{array}{rcl}
\sigma \vDash p & \Leftrightarrow & p \in V(s_0) \\
\sigma \vDash X\varphi & \Leftrightarrow & \sigma_1 \vDash \varphi \\
\sigma \vDash \psi U \varphi & \Leftrightarrow & \underline{\text{存在}} \text{ 一个 } k \text{ 使得 } \sigma_k \vDash \varphi \text{ 并且对所有 } j < k, \sigma_j \vDash \psi
\end{array}
$$

- 回忆: $\varphi W \psi := (\varphi U \psi) \vee G\varphi$
- $G(req_1 \wedge req_2 \rightarrow (\neg own_2\ W(own_2\ W(\neg own_2\ W\ own_1))))$ 什么时候为真?

LTL 的语义

LTL 公式的语义定义在 (无穷) 序列 $\sigma = s_0 s_1 s_2 \ldots$ 上 (每个点上有基本命题赋值 V), 令 σ_k 为从 σ 的 k 位重新开始的 σ 的子串:

$$
\begin{aligned}
\sigma \vDash p &\Leftrightarrow p \in V(s_0) \\
\sigma \vDash X\varphi &\Leftrightarrow \sigma_1 \vDash \varphi \\
\sigma \vDash \psi U \varphi &\Leftrightarrow \underline{\text{存在}} \text{ 一个 } k \text{ 使得 } \sigma_k \vDash \varphi \text{ 并且对所有 } j < k, \sigma_j \vDash \psi
\end{aligned}
$$

- 回忆: $\varphi W \psi := (\varphi U \psi) \vee G\varphi$
- $G(req_1 \wedge req_2 \rightarrow (\neg own_2\ W(own_2\ W(\neg own_2\ W\ own_1))))$ 什么时候为真?

据此也可以定义克里普克模型 (看成计算机的状态转移模型) 上公式的真值:

$$
\mathcal{M}, s \vDash \varphi \Leftrightarrow \text{从 } s \text{ 出发的}\underline{\text{所有}}\mathcal{M} \text{ 中的路径 } \sigma \text{ 都有} \sigma \vDash \varphi
$$

LTL 的语义

LTL 公式的语义定义在 (无穷) 序列 $\sigma = s_0 s_1 s_2 \ldots$ 上 (每个点上有基本命题赋值 V), 令 σ_k 为从 σ 的 k 位重新开始的 σ 的子串:

$$
\begin{array}{rcl}
\sigma \vDash p & \Leftrightarrow & p \in V(s_0) \\
\sigma \vDash X\varphi & \Leftrightarrow & \sigma_1 \vDash \varphi \\
\sigma \vDash \psi U \varphi & \Leftrightarrow & \underline{\text{存在}} \text{ 一个 } k \text{ 使得 } \sigma_k \vDash \varphi \text{ 并且对所有 } j < k, \sigma_j \vDash \psi
\end{array}
$$

- 回忆: $\varphi W \psi := (\varphi U \psi) \vee G\varphi$
- $G(req_1 \wedge req_2 \rightarrow (\neg own_2 \; W(own_2 \; W(\neg own_2 \; W \; own_1))))$ 什么时候为真?

据此也可以定义克里普克模型 (看成计算机的状态转移模型) 上公式的真值:

$$\mathcal{M}, s \vDash \varphi \quad \Leftrightarrow \quad \text{从 } s \text{ 出发的}\underline{\text{所有}}\mathcal{M} \text{ 中的路径 } \sigma \text{ 都有} \sigma \vDash \varphi$$

LTL 在模型 $\mathcal{M}$ 上表述的都是某种安全性 (saefty property): 所有执行序列都怎么怎么样...

分叉时间时态逻辑 CTL (COMPUTATION TREE LOGIC)

语言包括: $EX\varphi$, $EG\varphi$, $\psi EU\varphi$,

分叉时间时态逻辑 CTL (COMPUTATION TREE LOGIC)

语言包括: $EX\varphi$, $EG\varphi$, $\psi EU\varphi$, $AF\varphi := \neg EG\neg\varphi$,
$AG\varphi := \neg EF\neg\varphi := \neg(\top EU\neg\varphi)$.

分叉时间时态逻辑 CTL (COMPUTATION TREE LOGIC)

语言包括: $EX\varphi$, $EG\varphi$, $\psi EU\varphi$, $AF\varphi := \neg EG\neg\varphi$,

$AG\varphi := \neg EF\neg\varphi := \neg(\top EU\neg\varphi)$.

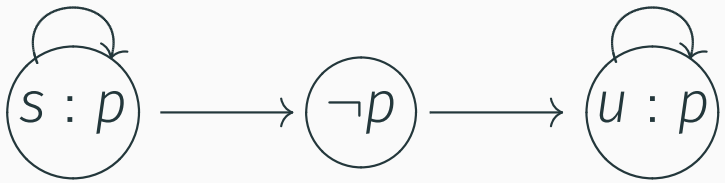
$\mathcal{M}, s \Vdash p$	$\Leftrightarrow$	$p \in V(s)$
$\mathcal{M}, s \Vdash EX\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得$s_1 \Vdash \varphi$
$\mathcal{M}, s \Vdash EG\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得 对所有路径上的点$s_i \Vdash \varphi$
$\mathcal{M}, s \Vdash \psi EU\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得 存在 一个 k 使得 $s_k \Vdash \varphi$ 并且对所有 $j < k, s_j \Vdash \psi$

分叉时间时态逻辑 CTL (COMPUTATION TREE LOGIC)

语言包括: $EX\varphi$, $EG\varphi$, $\psi EU\varphi$, $AF\varphi := \neg EG\neg\varphi$,

$AG\varphi := \neg EF\neg\varphi := \neg(\top EU\neg\varphi)$.

$$
\begin{array}{rcl}
\mathcal{M}, s \Vdash p & \Leftrightarrow & p \in V(s) \\
\mathcal{M}, s \Vdash EX\varphi & \Leftrightarrow & \text{存在一个从 s 出发的路径 } s_0 s_1 \ldots \text{ 使得} s_1 \Vdash \varphi \\
\mathcal{M}, s \Vdash EG\varphi & \Leftrightarrow & \text{存在一个从 s 出发的路径 } s_0 s_1 \ldots \text{ 使得} \\
 & & \text{对所有路径上的点} s_i \Vdash \varphi \\
\mathcal{M}, s \Vdash \psi EU\varphi & \Leftrightarrow & \text{存在一个从 s 出发的路径 } s_0 s_1 \ldots \text{ 使得} \\
 & & \underline{\text{存在}} \text{ 一个 } k \text{ 使得 } s_k \Vdash \varphi \text{ 并且对所有 } j < k, s_j \Vdash \psi
\end{array}
$$

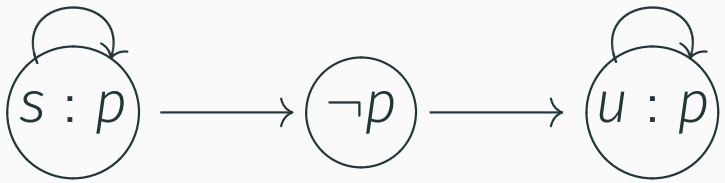


分叉时间时态逻辑 CTL (COMPUTATION TREE LOGIC)

语言包括: $EX\varphi$, $EG\varphi$, $\psi EU\varphi$, $AF\varphi := \neg EG\neg\varphi$,

$AG\varphi := \neg EF\neg\varphi := \neg(\top EU\neg\varphi)$.

$\mathcal{M}, s \Vdash p$	$\Leftrightarrow$	$p \in V(s)$
$\mathcal{M}, s \Vdash EX\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得$s_1 \Vdash \varphi$
$\mathcal{M}, s \Vdash EG\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得 对所有路径上的点$s_i \Vdash \varphi$
$\mathcal{M}, s \Vdash \psi EU\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得 存在 一个 k 使得 $s_k \Vdash \varphi$ 并且对所有 $j < k, s_j \Vdash \psi$



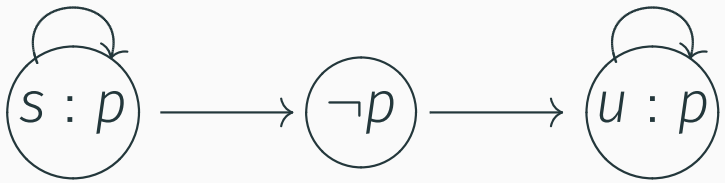
$\mathcal{M}, s \vDash FGp$ (LTL 公式),

分叉时间时态逻辑 CTL (COMPUTATION TREE LOGIC)

语言包括: $EX\varphi$, $EG\varphi$, $\psi EU\varphi$, $AF\varphi := \neg EG\neg\varphi$,

$AG\varphi := \neg EF\neg\varphi := \neg(\top EU\neg\varphi)$.

$\mathcal{M}, s \Vdash p$	$\Leftrightarrow$	$p \in V(s)$
$\mathcal{M}, s \Vdash EX\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得$s_1 \Vdash \varphi$
$\mathcal{M}, s \Vdash EG\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得 对所有路径上的点$s_i \Vdash \varphi$
$\mathcal{M}, s \Vdash \psi EU\varphi$	$\Leftrightarrow$	存在一个从 s 出发的路径 $s_0 s_1 \ldots$ 使得 存在 一个 k 使得 $s_k \Vdash \varphi$ 并且对所有 $j < k, s_j \Vdash \psi$



$\mathcal{M}, s \vDash FGp$ (LTL 公式), 但是 $\mathcal{M}, s \nvDash \neg AF\, AGp$ (CTL) 公式 (为什么?).

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题?

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低.

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低. “A picture is worth a thousand words”. (自拍胜千言?)

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低. “A picture is worth a thousand words”. (自拍胜千言?)

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低. “A picture is worth a thousand words”. (自拍胜千言?)

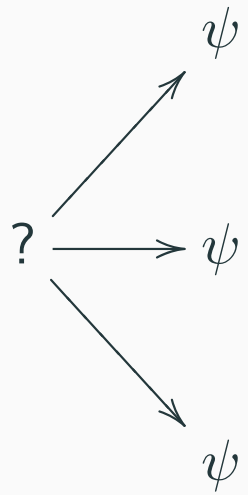
比如 CTL 公式 $AF\varphi$ 的 labelling 算法, 有穷多步可停止.

模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低. “A picture is worth a thousand words”. (自拍胜千言?)

比如 CTL 公式 $AF\varphi$ 的 labelling 算法, 有穷多步可停止.

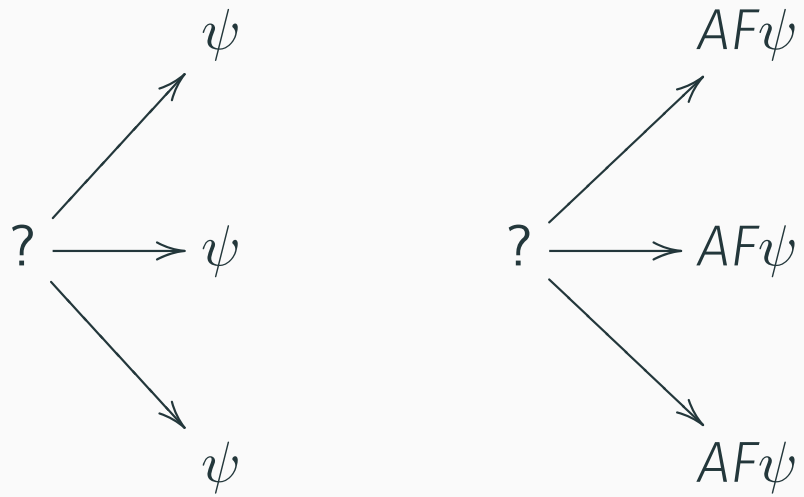


模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低. “A picture is worth a thousand words”. (自拍胜千言?)

比如 CTL 公式 $AF\varphi$ 的 labelling 算法, 有穷多步可停止.

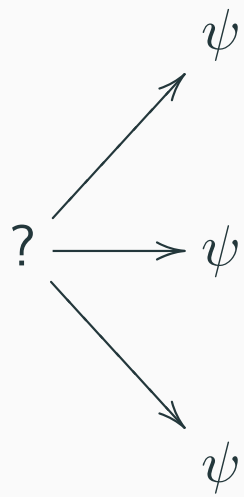


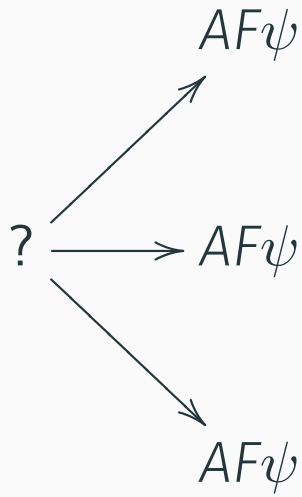
模型检测 (MODEL CHECKING)

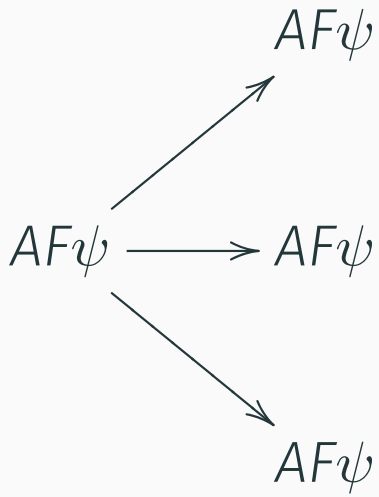
基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低. “A picture is worth a thousand words”. (自拍胜千言?)

比如 CTL 公式 $AF\varphi$ 的 labelling 算法, 有穷多步可停止.





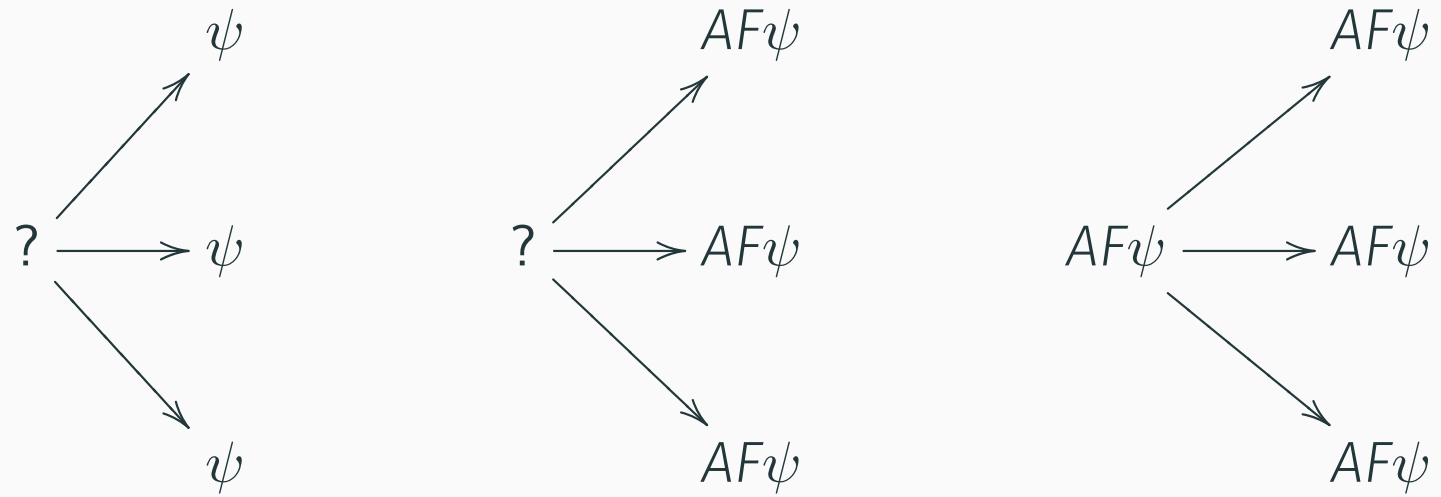


模型检测 (MODEL CHECKING)

基本的问题就是判断是否 $\mathcal{M}, s \vDash \varphi$ (不成立则指出模型里哪不对)

- 为什么这是个问题? 特定模态逻辑的语义定义没那么简单.
- 为什么关心这个问题? 程序或者硬件可以抽象成数学模型, 不需要先用一大堆 (甚至无穷多) 公式刻画它再做推理. 同一个逻辑的模型检测一般比定理证明或可满足性判断计算复杂度更低. “A picture is worth a thousand words”. (自拍胜千言?)

比如 CTL 公式 $AF\varphi$ 的 labelling 算法, 有穷多步可停止.



有 φ 标 $AF\varphi$, 后继都有 $AF\varphi$ 则也标 $AF\varphi$, 重复上一步操作至停止.

07 年图灵奖

颁给 Clarke, Emerson 以及 Sifakis 基于他们对模型检测的开创性贡献.

07 年图灵奖

颁给 Clarke, Emerson 以及 Sifakis 基于他们对模型检测的开创性贡献.

“Model Checking started as an academic research idea. The continuing research of Clarke, Emerson, and Sifakis as well as others in the international research community over the last 27 years led to the creation of new logics, as well as new algorithms and surprising theoretical results. This in turn has stimulated the creation of many Model Checking tools by both academic and industrial teams, resulting in the widespread industrial use of Model Checking. ”

模型检测最大的敌人

状态空间爆炸 (state space explosion) 280 个布尔变量 (比如灯的开关) 的所有状态组合就超过了整个可观测宇宙的原子数量.

模型检测最大的敌人

状态空间爆炸 (state space explosion) 280 个布尔变量 (比如灯的开关) 的所有状态组合就超过了整个可观测宇宙的原子数量.

办法:

- Symbolic model checking: 更有效的表示模型

模型检测最大的敌人

状态空间爆炸 (state space explosion) 280 个布尔变量 (比如灯的开关) 的所有状态组合就超过了整个可观测宇宙的原子数量.

办法:

- Symbolic model checking: 更有效的表示模型
- Abstraction: 大模型变小扔掉不用的信息.

模型检测最大的敌人

状态空间爆炸 (state space explosion) 280 个布尔变量 (比如灯的开关) 的所有状态组合就超过了整个可观测宇宙的原子数量.

办法:

- Symbolic model checking: 更有效的表示模型
- Abstraction: 大模型变小扔掉不用的信息.
- Partial order reduction: 去除等价的情况.

模型检测最大的敌人

状态空间爆炸 (state space explosion) 280 个布尔变量 (比如灯的开关) 的所有状态组合就超过了整个可观测宇宙的原子数量.

办法:

- Symbolic model checking: 更有效的表示模型
- Abstraction: 大模型变小扔掉不用的信息.
- Partial order reduction: 去除等价的情况.
- Composition: 拆分问题.
- ...

Now let’s welcome Malvin Gattinger.

Now let's welcome Malvin Gattinger. A talk by Malvin (Next Tuesday 15:10pm Yi Jiao 105.):