Hennessy-Milner Logic[2]

- Abbiamo già detto che il linguaggio concorrente visto finora (CCS) si presta al *single-language approach*, ovvero è adatto a scrivere sia la specifica che l'implementazione di un sistema.

- Abbiamo visto che provare

$$Impl \approx Spec$$

  è un modo naturale per verificare che un'implementazione $Impl$ rispetti la specifica $Spec$.

- Però, così facendo, siamo implicitamente obbligati a verificare l'implementazione per intero.

- Quando si lavora con sistemi molto complessi è sicuramente auspicabile avere un criterio per verificare che l'implementazione soddisfi esclusivamente determinate proprietà.

[2]Tratto dalle note del corso "Semantics and Verification" tenuto dal Prof. Luca Aceto ad Aalborg, Danimarca.

- Ecco, ad esempio, alcune proprietà che potremmo essere interessati a verificare:

    1. Il sistema P è in grado di eseguire un'azione $\xrightarrow{a}$ (subito).
    2. Il sistema P è in grado di eseguire un azione $\xrightarrow{a}$ ma non un'azione $\xrightarrow{b}$.
    3. Il sistema P è in grado di eseguire un'azione a che lo condurrà in uno stato in cui non è possibile eseguire alcuna azione b.
    4. Se il sistema P esegue un'azione a allora raggiungerà sempre un qualche stato in cui non è possibile eseguire un'azione b.
    5. Se il sistema P esegue un'azione a allora raggiungerà sempre un qualche stato in cui ogni qual volta che viene eseguita un'azione b si raggiungerà un terzo stato in cui è possibile eseguire l'azione c ma non d.
    6. etc...

- Modellare tali proprietà utilizzando $\approx$ è sicuramente innaturale, e tutt'altro che facile.

- È invece molto più ragionevole definire un qualche formalismo (o linguaggio) per esprimere in maniera adeguata le proprietà che intendiamo verificare.

- Tale linguaggio deve avere una sintassi ed una semantica formale, così da poter essere implementato su un computer.

- Inoltre deve possedere degli algoritmi per poter verificare se un processo soddisfi o meno una certa proprietà.

- Tale approccio di verifica della specifica di un sistema viene spesso chiamato *model checking*.

- Quando si fa model checking si usano di solito due linguaggi differenti: uno per descrivere la specifica, ed uno per descriverere i sistemi veri e propri (ovvero l'implementazione).

- Per esempio possiamo usare CCS (il nostro linguaggio per processi concorrenti) per descrivere l'implementazione ed una qualche forma di *logica* (ovvero un linguaggio per descrivere proprietà) per dare la specifica.

- La logica che ci accingiamo a presentare è universalmente conosciuta come *Hennessy-Milner Logic* (abbreviata HML).

Definizione 69 (Sintassi) *L'insieme di formule à la Hennessy-Milner su un insieme di azioni Act è data dalla seguente sintassi:*

$$F \ ::= \ \mathsf{t} \ \mid \ \mathsf{f} \ \mid \ F \wedge G \ \mid \ F \vee G \ \mid \ \langle a \rangle F \ \mid \ [a]F$$

dove $a \in Act$. *Per* $A = \{a_1, \ldots, a_n\} \subseteq Act$ *useremo l'abbreviazione* $\langle A \rangle F$ *per la formula* $\langle a_1 \rangle F \vee \ldots \vee \langle a_n \rangle F$; *similmente, scriveremo* $[A]F$ *per abbreviare* $[a_1]F \wedge \ldots \wedge [a_n]F$. *(Se* $A = \emptyset$ *allora* $\langle A \rangle F = \mathsf{f}$ *e* $[A]F = \mathsf{t}$*).*

- Indicheremo con $\mathcal{F}$ l'insieme delle formule à la Hennessy-Milner che possono essere derivate attraverso la sintassi della Definizione 69.

- Il significato (o *semantica*) di una formula del linguaggio $\mathcal{F}$ è dato dall'insieme dei processi CCS che soddisfano quella formula.

- Intuitivamente,

  – Tutti i processi soddisfano t;

- – Nessun processo soddisfa f;
  - – Un processo soddisfa la formula $F \wedge G$ se e solo se soddisfa sia F che G;
  - – Un processo soddisfa la formula $F \vee G$ se e solo se soddisfa o F oppure G;
  - – Un processo P soddisfa $\langle a \rangle F$, per $a \in Act$, se e solo se esiste P' tale che $P \xrightarrow{a} P'$ e P' soddisfa F;
  - – Un processo P soddisfa $[a]F$, per $a \in Act$, se e solo per ogni P', tale che $P \xrightarrow{a} P'$, P' soddisfa F.

- Quindi, intuitivamente, una formula del tipo $\langle a \rangle F$ denota la *possibilità* di eseguire un'azione "a" raggiungendo uno stato (ovvero un processo) che soddisfi la proprietà F.

- Se invece un processo soddisfa una proprietà del tipo $[a]F$ allora qualsiasi sia l'azione a eseguita dal processo si raggiunge un altro processo che deve *necessariamente* avere la proprietà F.

- Logiche che parlano di *necessità* e *possibilità* sono dette **logiche modali**.

- La *semantica* del linguaggio delle formule $\mathcal{F}$ è data rispetto ad un LTS; nel nostro caso consideremo l'LTS per processi concorrenti che abbiamo già definito.

- Scriveremo $[\![F]\!]$ per rappresentare la semantica (*denotazionale*) della formula F, ovvero l'insieme dei processi in $\mathcal{P}$ che soddisfano la proprietà F.

Definizione 70 (Semantica di $\mathcal{F}$) *Definiamo $[\![F]\!] \subseteq \mathcal{P}$ per induzione sulla struttura di $F \in \mathcal{F}$:*

$$
\begin{array}{lclclcl}
[\![\mathsf{t}]\!] & \stackrel{def}{=} & \mathcal{P} & \qquad & [\![F \vee G]\!] & \stackrel{def}{=} & [\![F]\!] \cup [\![G]\!] \\
[\![\mathsf{f}]\!] & \stackrel{def}{=} & \emptyset & \qquad & [\![F \wedge G]\!] & \stackrel{def}{=} & [\![F]\!] \cap [\![G]\!]
\end{array}
$$

$$[\![\langle a \rangle F]\!] \stackrel{def}{=} \{P \in \mathcal{P} : \exists P'. P \xrightarrow{a} P' \ e \ P' \in [\![F]\!]\}$$

$$[\![[a]F]\!] \stackrel{def}{=} \{P \in \mathcal{P} : \forall P'. P \xrightarrow{a} P' \ implica \ P' \in [\![F]\!]\}$$

Notazione: *Scriveremo $P \vdash F$ (P soddisfa F) se e solo se $P \in [\![F]\!]$.*

Rivediamo allora come sia possibile esprimere le proprietà menzionate in precedenza utilizzando la HML.

1. Il sistema P è in grado di eseguire un'azione $\xrightarrow{a}$ (subito):

$$P \vdash \langle a \rangle \mathsf{t}$$

2. Il sistema P è in grado di eseguire un azione $\xrightarrow{a}$ ma non un'azione $\xrightarrow{b}$:

$$P \vdash \langle a \rangle \mathsf{t} \ \wedge \ [b]\mathsf{f}$$

3. Il sistema P è in grado di eseguire un'azione a che lo condurrà in uno stato in cui non è possibile eseguire alcuna azione b:

$$P \vdash \langle a \rangle [b]\mathsf{f}$$

4. Se il sistema P esegue un'azione a allora raggiungerà sempre un qualche stato in cui è possibile eseguire un'azione b:

$$P \vdash [a]\langle b \rangle \mathsf{t}$$

5. Ogni qual volta il sistema P esegue un'azione a raggiungerà un qualche stato in cui ogni qual volta si esegue un'azione b si raggiungerà un terzo stato in cui è possibile eseguire l'azione c ma non d:
$$P \vdash [a][b](\langle c\rangle \mathsf{t} \ \wedge \ [d]\mathsf{f})$$

Domanda: Cosa esprimono le formule $\langle a\rangle\mathsf{f}$ e $[a]\mathsf{t}$?

Esercizio 71 *(1) Trovare una formula in $\mathcal{F}$ che sia soddisfatta da $a.b.\mathbf{0}+a.c.\mathbf{0}$ ma non da $a.(b.\mathbf{0}+c.\mathbf{0})$. (2) Trovare una formula in $\mathcal{F}$ che sia soddisfatta da $a.(b.c.\mathbf{0}+b.d.\mathbf{0})$ ma non da $a.b.c.\mathbf{0}+a.b.d.\mathbf{0}$.*

Si noti che è possibile fornire una caratterizzazione alternativa della relazione di soddisfacibilità $\vdash$ data nella Definizione 70. Tale formulazione è data per induzione sulla struttura delle formule F (*structural induction*):

- $P \vdash \mathsf{t}$, per ogni P;
- $P \vdash \mathsf{f}$, per nessun P;

- $P \vdash F \wedge G$ sse $P \vdash F$ e $P \vdash G$;
- $P \vdash F \vee G$ sse $P \vdash F$ oppure $F \vdash G$;
- $P \vdash \langle a \rangle F$ sse $P \xrightarrow{a} P'$ per qualche P' tale che $P' \vdash F$;
- $P \vdash [a]F$ sse $P' \vdash F$, per ogni P' tale che $P \xrightarrow{a} P'$.

Negazione

- Si noti che la negazione non è uno dei costrutti della nostra logica.
- Questo perchè la logica è chiusa rispetto alla negazione, ovvero per ogni formula $F \in \mathcal{F}$ esiste una formula F^{C} equivalente alla negazione di F.
- Le formule F^{C} sono definite per induzione strutturale su F:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
\mathsf{t}^{\mathsf{C}} & \stackrel{\text{def}}{=} & \mathsf{f} & (F \vee G)^{\mathsf{C}} & \stackrel{\text{def}}{=} & F^{\mathsf{C}} \wedge G^{\mathsf{C}} \\
\mathsf{f}^{\mathsf{C}} & \stackrel{\text{def}}{=} & \mathsf{t} & (\langle a \rangle F)^{\mathsf{C}} & \stackrel{\text{def}}{=} & [a]F^{\mathsf{C}} \\
(F \wedge G)^{\mathsf{C}} & \stackrel{\text{def}}{=} & F^{\mathsf{C}} \vee G^{\mathsf{C}} & ([a]F)^{\mathsf{C}} & \stackrel{\text{def}}{=} & \langle a \rangle F^{\mathsf{C}}
\end{array}
$$

- **Proposizione 72** *Dato un LTS* $= (\mathsf{Proc}, Act, \{ \xrightarrow{a} | \ a \in Act\})$. *Allora, per ogni formula* $F \in \mathcal{F}$, *vale che* $[\![F^{\mathsf{c}}]\!] = \mathsf{Proc} \setminus [\![F]\!]$.

- Una conseguenza della proposizione precedente è che per ogni processo P e formula F vale esattamente una (ed una sola) tra $P \vdash F$ e $P \vdash F^{\mathsf{c}}$. Infatti il processo P sarà contenuto o in $[\![F]\!]$ oppure nel suo complementare $[\![F^{\mathsf{c}}]\!]$.

Equivalenza forte vs HML

- Nell'Esercizio 71 veniva chiesto di trovare delle formule che distinguessero processi che sapevamo non essere equivalenti in senso forte.

- Come ulteriore esempio si considerino i processi:

$$\begin{aligned} A &\stackrel{\text{def}}{=} a.A + a.\mathbf{0} \\ B &\stackrel{\text{def}}{=} a.a.B + a.\mathbf{0} \end{aligned}$$

- I due processi non sono equivalenti in senso forte.

- Rispondendo alla domanda precedente siamo obbligati a fare un'analisi dalla quale possiamo concludere che la formula

$$F \stackrel{\text{def}}{=} \langle a\rangle\langle a\rangle[a]\mathsf{f}$$

è in grado di distinguere i due processi, poichè $A \vdash F$ mentre $B \not\vdash F$.

- Quindi, una volta ancora, dati due processi non equivalenti siamo riusciti a trovare una formula che li distingua

- **Domanda1:** Questa regola vale in generale?

- **Domanda2:** Cosa possiamo dire su due processi equivalenti in senso forte? Soddisfano le stesse formule?

- A queste domande risponde un famoso teorema di Hennessy e Milner.

- Prima però di enunciare il teorema abbiamo bisogno di definire una particicolare classe di processi.

Definizione 73 (Processi Image Finite) *Un processo P è detto* image finite *sse l'insieme $\{P' : P \xrightarrow{a} P'\}$ è finito, per ogni azione a. Un LTS è image finite se lo è ogni suo processo.*

- Ad esempio il processo

$$A \stackrel{\text{def}}{=} a.\mathbf{0} \mid A$$

Non è image finite in quanto, per ogni $n \geq 0$, vale

$$A \xrightarrow{a} \underbrace{a.\mathbf{0} \mid \ldots \mid a.\mathbf{0}}_{\text{n volte}} \mid \mathbf{0} \mid A$$

- Un altro processo non image finite è

$$A^{<\omega} \stackrel{\text{def}}{=} \sum_{i \geq 0} a^i.\mathbf{0} = \mathbf{0} + a.\mathbf{0} + a.a.\mathbf{0} + a.a.a.\mathbf{0} + \ldots$$

- Siamo pronti allora a provare il Teorema principale dell'HML.

Teorema 74 (HML vs $\sim$) *Sia* $(\mathsf{Proc}, Act, \{\xrightarrow{a} \mid a \in Act\})$ *un LTS image finite, e siano* $P, Q \in \mathsf{Proc}$. *Allora* $P \sim Q$ *sse* P *e* Q *soddisfano esattamente le stesse formule in* $\mathcal{F}$.

- In realtà l'implicazione da sinistra verso destra vale per tutti i processi, non solo per quelli image finite.

- Questo teorema ha varie applicazioni. In particolar modo applicandolo all'inverso ne deriva che:

  - Dati due processi P e Q. Se esiste una formula F che li distingue, cioè tale che $P \vdash F$ e $Q \nvdash F$ (o viceversa), allora $P \not\sim Q$.
  - Se due processi P e Q, image finite, non sono equivalenti in senso forte allora esiste sicuramente una formula F che li distingue.

- Al fine di capire il motivo per cui il teorema precedente vale solo per processi image finite, quelli di voi che amano le cose interessanti possono provare a fare il seguente esercizio.

Esercizio 75 *Sia* $A^\omega \stackrel{def}{=} a.A^\omega$. *Mostrare che i processi* $A^{<\omega}$ *e* $A^\omega + A^{<\omega}$

1. *Non sono equivalenti in senso forte (e neppure in senso debole)*
2. *Soddisfano le stesse formule in* $\mathcal{F}$

Suggerimento: Per il secondo punto si usi l'induzione strutturale sulle formule in $\mathcal{F}$. *A tal fine è utile provare un risultato intermedio:*

$A^\omega \vdash F$ *per* $F \in \mathcal{F}$ *sse* $a^i \vdash F$ *dove* i *è la profondità modale ovvero il numero massimo di annidamenti degli operatori modali.*

Equivalenza debole vs HML

- La HML può essere facilmente estesa per lavorare con transizioni deboli, $\stackrel{a}{\Longrightarrow}$, e quindi con l'equivalenza debole $\approx$.

- La WHML (Weak HML) si differisce dalla HML per gli operatori modali che adesso sono definiti in termini delle azioni deboli, ovvero:
  - Un processo P soddisfa $\langle\langle a \rangle\rangle F$, per $a \in Act$, se e solo se esiste P' tale che $P \stackrel{a}{\Longrightarrow} P'$ e P' soddisfa F;
  - Un processo P soddisfa $\langle\langle \rangle\rangle F$ se e solo se esiste P' tale che $P \Rightarrow P'$ e P' soddisfa F;
  - Un processo P soddisfa $[\![a]\!]F$, per $a \in Act$, se e solo per ogni P', tale che $P \stackrel{a}{\Longrightarrow} P'$, P' soddisfa F.
  - Un processo P soddisfa $[\![]\!]F$ se e solo per ogni P', tale che $P \Rightarrow P'$, P' soddisfa F.
- Chiamiamo $\mathcal{F}^{\mathrm{W}}$ l'insieme di tutte le formule della WHML.
- Formalmente la semantica denotazione $[\![F]\!]$, per $F \in \mathcal{F}^{\mathrm{W}}$, degli

operatori modali deboli è così definita:

$$
\begin{array}{rcl}
[\![\langle\!\langle a \rangle\!\rangle F]\!] & \stackrel{\text{def}}{=} & \{P \in \mathcal{P} : \exists P'. P \stackrel{a}{\Longrightarrow} P' \text{ and } P' \in [\![F]\!]\} \\
[\![\langle\!\langle \rangle\!\rangle F]\!] & \stackrel{\text{def}}{=} & \{P \in \mathcal{P} : \exists P'. P \Rightarrow P' \text{ and } P' \in [\![F]\!]\} \\
[\![[\![a]\!] F]\!] & \stackrel{\text{def}}{=} & \{P \in \mathcal{P} : \forall P'. P \stackrel{a}{\Longrightarrow} P' \text{ implies } P' \in [\![F]\!]\} \\
[\![[\![]\!] F]\!] & \stackrel{\text{def}}{=} & \{P \in \mathcal{P} : \forall P'. P \Rightarrow P' \text{ implies } P' \in [\![F]\!]\}
\end{array}
$$

- La semantica degli altri operatori resta invariata.

- Allora, adattando la definizione di image-finite in senso debole abbiamo:

  Definizione 76 (Processi Image Finite in senso debole) *Un processo P è detto* image finite in senso debole *sse l'insieme* $\{P' : P \stackrel{a}{\Longrightarrow} P'\}$ *è finito per ogni azione a. Un LTS è image finite se lo è ogni suo processo.*

- Il risultato di Hennessy e Milner si estende nel seguente modo:

Teorema 77 (WHML vs $\approx$) *Sia* $(\mathsf{Proc}, Act, \{\xrightarrow{a} : a \in Act\})$ *un LTS image finite in senso debole, e siano* $P, Q \in \mathsf{Proc}$*. Allora* $P \approx Q$ *sse* P *e* Q *soddisfano esattamente le stesse formule in* $\mathcal{F}^{\mathrm{w}}$*.*

HML e comportamenti infiniti

- Una formula HML (o anche WHML) consente di descrivere solamente *comportamenti finiti*; ovvero individuati da un numero finito di passi corripondenti alla profondità della formula.

- Quanto si possa esplorare del comportamento di un processo attraverso una formula dipende dalla *profondità modale* della formula.

- Ad esempio verificare se un processo soddisfi la formula

$$[a]\langle a\rangle \mathsf{f} \ \wedge\ \langle b\rangle \mathsf{t}$$

  o meno richiede un'analisi delle transizioni del processo di lunghezza due.

- Comunque, molte proprietà importanti come ad esempio *safety* e *liveness* esprimono comportamenti che non si esauriscono in un numero finito di passi.

- Ad esempio, con $Inv(F)$ intenderemo che la proprietà F è un invariante, ovvero che è valida in ogni momento della computazione.

- Mentre, con $Pos(F)$ intenderemo che la proprietà F ha la possibilità di essere vera ad un certo momento della computazione.

- Ad esempio, si consideri il processo

$$A \stackrel{\text{def}}{=} a.A + b.A$$

  tale processo ha una *proprietà invariante*, ovvero ad ogni istante della computazione la proprietà $\langle a \rangle \texttt{t}$ è vera, cioè il processo può sempre eseguire un'azione "a".

- Come facciamo ad esprimere il fatto che tale proprietà vale sempre, cioè ad ogni istante della computazione?

- **Attraverso la HML non è possibile**. Dobbiamo perciò estendere la logica. Ad esempio, con *congiunzioni e disgiunzioni infinite*.
- La proprietà di prima potremmo allora definirla come

$$Inv(\langle a \rangle \mathsf{t}) \stackrel{\text{def}}{=} \langle a \rangle \mathsf{t} \wedge [a,b]\langle a \rangle \mathsf{t} \wedge [a,b][a,b]\langle a \rangle \mathsf{t} \wedge \ldots = \bigwedge_{i=0}^{\infty} [a,b]^i \langle a \rangle \mathsf{t}$$

- Consideriamo adesso un altro esempio molto semplice:

$$C \stackrel{\text{def}}{=} a.b.a.C + b.\mathbf{0}$$

- Tale processo ha la possibilità di terminare durante la computazione, sebbene non immediatamente.
- In questo caso usando le disgiunzioni infinite potremmo esprimere la propietà come:

$$Pos([a,b]\mathsf{f}) \stackrel{\text{def}}{=} [a,b]\mathsf{f} \vee \langle a,b \rangle [a,b]\mathsf{f} \vee \langle a,b \rangle^2 [a,b]\mathsf{f} \vee \ldots = \bigvee_{i=0}^{\infty} \langle a,b \rangle^i [a,b]\mathsf{f}$$

- Sebbene sia teoricamente possibile estendere la HML con congiunzioni e disgiunzioni infinite, le formule infinite non sono molto agevoli da usare.

- Al loro posto invece, conviene introdurre nella logica le definizioni ricorsive (un po' come abbiamo fatto per i processi).

- Ad esempio, sia Act l'insieme di azioni possibili, la proprietà $Inv(F)$, per una data formula F, possiamo esprimerla come la *soluzione massimale* dell'equazione

$$X \overset{\max}{=} F \wedge [Act]X$$

- Mentre $Pos(F)$ come la *soluzione minimale* dell'equazione

$$Y \overset{\min}{=} F \vee \langle Act \rangle Y$$

- Una soluzione G è massimale se $[\![G]\!]$ è massimale. Intuitivamente si usano soluzioni massimali per quelle proprietà che valgono sempre, cioè per cui non esiste una computazioni finita che la neghi. È questo il caso di $Inv(F)$.

- Una soluzione G è minimale se $[\![G]\!]$ è minimale. Intuitivamente si usano soluzioni minimali per quelle proprietà che valgono in processi che hanno una computazione finita che verifica la proprietà. Come ad esempio $Pos(F)$.

- Approfondimenti sulla HML con definizioni ricorsive possono essere trovate sugli appunti del corso del Prof. Aceto.