Introduzione

Dovendo progettare un sistema che risolva certi requisiti l'obiettivo che ci si pone è avere uno "strumento" (=linguaggio) che consenta una serie di attività:

- Specifica dei requisiti del sistema
- Verifica delle proprietà del sistema
- Simulazione del sistema
- Test del sistema
- "Esecuzione" del sistema

Metodi Formali di Specifica

Ing. Pierfrancesco Bellini

Linguaggio di Specifica

- Formale (sintassi e semantica precise)
  - necessario per la verifica di proprietà
  - sistemi "safety-critical"
- Espressivo
- Intuitivo/Usabile
  - linguaggio visuale
  - tool di supporto alla specifica

Esempio: specifica di requisiti

- Se sono nello stato A e accade B allora vado nello stato S
- Non si possono estrarre messaggi dal buffer quando questo è vuoto
- Il buffer può contenere al massimo N messaggi
- Quando la condizione C diventa vera allora si esegue D entro s secondi
- Il processo P deve essere eseguito ogni t secondi e deve produrre risultatientro t_2 sec.

Esempio: Verifica

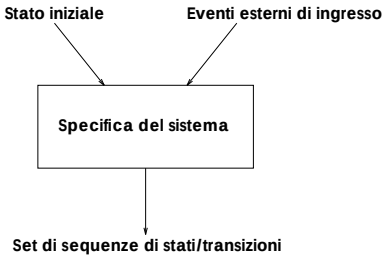
Verifica di consistenza e completezza:

- Consistenza: è ogni parte della specifica non in conflitto con altre parti?
- Completezza: il sistema è descritto in modo completo? Il modello permette di verificare la completezza? ogni possibile dettaglio è stato specificato?

Esempio: test della specifica

- Se sono nello stato S, può accadere la sequenza di eventi $< e_1, e_2, \ldots, e_n >$?
- Se sono nello stato A, l'evento B accade in un tempo compreso fra t_1 e t_2?

Esempio: simulazione del sistema



Generazione di sequenze di stati ed eventi che sono possibili per il sistema specificato, da utilizzare per controllare che eventuali comportamenti non desiderati non si verifichino.

È ovviamente un metodo non esaustivo di verifica, ma può sempre essere utile (in pratica, si tratta di un "debugging" della specifica).

Permette di lavorare con un prototipo del sistema che si sta progettando fin dalla fase di specifica evidenziando ulteriori requisiti, errori nei requisiti specificati e fraintendimenti fra progettista e utente.

Esempio: verifica di proprietà

- Il sistema trasmette messaggi da Msg_in a Msg_out
- Il sistema non duplica i messaggi
- Il sistema non crea nuovi messaggi
- Il sistema non riordina i messaggi
- Se invio un messaggio, prima o poi il sistema chiede l'invio di un nuovo messaggio

Possibili Approcci

- Metodi operazionali
  - modelli a stati/transizioni
- Metodi descrittivi/denotazionali
  - Modelli algebrici
  - Modelli logici
- Metodi duali

Esempio: esecuzione del sistema

- Trasformazione della specifica attraverso passaggi successivi in un sistema eseguibile
- Traduzione della specifica in un linguaggio eseguibile (e.g., operazionale)
- Interpretazione della specifica in tempo reale.

Metodi descrittivi

Metodi algebrici

- Sono basati sul concetto di Abstract Data Type
- Un Abstract Data Type viene specificato elencando gli operatori che possono agire sul dato stesso
- Ciascun operatore è definito da un insieme di assiomi che ne descrivono il comportamento e le proprietà fondamentali
- Esempio: Larch

Lista : **trait**
introduces
new :→ List
add : List, Elem → List
del : List, Elem → List
_ ∈ _ : Elem, List → Bool
asserts ∀ e, f : Elem, l : List
$\neg(e \in \text{new})$
$e \in \text{add}(l, f) \equiv e = f \vee e \in l$
$\text{del}(\text{add}(l, f), e) \equiv$ **if** $e = f$ **then** l **else** $\text{add}(\text{del}(l, e), f)$

Metodi operazionali

- Specifica del sistema descrivendone il comportamento in termini di stati e transizioni.
- Facilmente comprensibili perché descrivono il comportamento del sistema.
- La specifica è intrinsecamente eseguibile.
- Difficoltà nella estrazione di proprietà sintetiche: esplosione esponenziale dello spazio degli stati.
- Esempi: Macchine a stati (ed estensioni: OSDL, XST, STATECHART), Reti di Petri (ed estensioni: TPN, CTPN, SPN).

Metodi descrittivi

Metodi matematici/logici/algebrici

- Lo stato del sistema viene descritto in modo matematico
- Le operazioni vengono definite indicando quale parte dello stato viene interessata e quale relazione sussiste tra lo stato prima e dopo l'esecuzione dell'operazione
- Esempio: Z, VDM

Metodi descrittivi

Metodi logici

- Descrizione della storia dello stato del sistema tramite un insieme di assiomi che ne determinano le possibili evoluzioni
- Gli assiomi determinano le relazioni di precedenza tra eventi e l'abilitazione di transizioni in base allo stato del sistema
- Esempio: logiche temporali

Sommario

- Logica proposizionale e logica del primo ordine.
- Logiche temporali.
- Sistemi tempo-reale.
- Specifica di sistemi tempo-reale con logiche temporali.

Metodi duali

- Integrano i metodi descrittivi con i metodi operazionali
- Operano in genere per raffinamenti successivi, inizialmente specifica più descrittiva (astratta) verso specifica più operazionale (concreta)
- Esempio: logiche temporali eseguibili

Logica Proposizionale: Interpretazione

Definizione 1: Data una formula proposizionale A e l'insieme $\{p_1, p_2, \ldots, p_n\}$ di proposizioni atomiche che appaiono in A, una interpretazione è una funzione $v : \{p_1, p_2, \ldots, p_n\} \to \{\top, \bot\}$, cioè v assegna un *valore di verità* a ciascun atomo.

Conseguentemente v assegna un valore $\top$ o $\bot$ ad A secondo la definizione induttiva riportata in tabella:

A	$v(A_1)$	$v(A_2)$	$v(A)$
$\top$			$\top$
$\bot$			$\bot$
$\neg A_1$	$\top$		$\bot$
$\neg A_1$	$\bot$		$\top$
$A_1 \vee A_2$	$\bot$	$\bot$	$\bot$
$A_1 \vee A_2$	otherwise		$\top$

A	$v(A_1)$	$v(A_2)$	$v(A)$
$A_1 \wedge A_2$	$\top$	$\top$	$\top$
$A_1 \wedge A_2$	otherwise		$\bot$
$A_1 \to A_2$	$\top$	$\bot$	$\bot$
$A_1 \to A_2$	otherwise		$\top$
$A_1 \leftrightarrow A_2$	$v(A_1) = v(A_2)$		$\top$
$A_1 \leftrightarrow A_2$	$v(A_1) \neq v(A_2)$		$\bot$

Logica Proposizionale: operandi e operatori

Costanti logiche	$\top$	true
	$\bot$	false
Variabili logiche	$A_1, A_2, \ldots$	con valore $\top$ o $\bot$
Connettivi logici	$\vee$	or
	$\wedge$	and
	$\neg$	not
	$\to$	implica
	$\leftrightarrow$	se e solo se

Le tabelle di verità permettono di definire la semantica, cioè il "significato", delle espressioni logiche. In pratica per ogni possibile combinazione dei valori delle variabili logiche è possibile determinare il valore che assume la formula.

Logica Proposizionale: Procedure di decisione

Definizione 6: data una classe di formule U, un algoritmo è una procedura di decisione per U se, data una arbitraria formula A, l'algoritmo termina e restituisce la risposta "sì" se $A \in U$ e la risposta "no" se $A \notin U$.

Una procedura di decisione per la soddisfacibilità può essere utilizzata come procedura di decisione per la validità: per vedere se A è valida basta applicare la procedura di decisione per la soddisfacibilità a $\neg A$. Se $\neg A$ è soddisfacibile, allora $\not\models A$, altrimenti $\models A$.

Applicando le definizioni degli operatori logici (tabelle di verità) si ha una procedura di decisione per la soddisfacibilità: basta provare una dopo l'altra le interpretazioni possibili della formula finché se ne trova una che la rende vera. Se non se ne trovano, la formula è non soddisfacibile.

Logica Proposizionale: Soddisfacibilità e Validità

Definizione 2: Una formula proposizionale A si dice *soddisfacibile* se il suo valore è $\top$ in una *qualche* interpretazione. Una tale interpretazione si chiama *modello* per A.

Definizione 3: Una formula proposizionale A si dice *non soddisfacibile* se il suo valore è $\bot$ in *ogni* interpretazione, cioè se $\neg A$ è soddisfacibile.

Definizione 4: Una formula proposizionale A si dice *valida* se è soddisfacibile in *ogni* interpretazione. La formula A viene chiamata *tautologia*. Notazione: $\models A$.

Definizione 5: Una formula proposizionale A si dice *non valida* o *falsificabile* se il suo valore è $\bot$ in una *qualche* interpretazione.

Teorema 1: A è valida se e solo se $\neg A$ è non soddisfacibile. A è soddisfacibile se e solo se $\neg A$ è falsificabile.

Logica del Primo Ordine: Interpretazioni

Definizione 8: Sia A una formula con $\{p_1, \ldots, p_m\}$ predicati e $\{a_1, \ldots, a_k\}$ costanti che appaiono in A, allora una *interpretazione* $\mathcal{I}$ è una tripla

$$(D, \{R_1, \ldots, R_m\}, \{d_1, \ldots, d_k\})$$

dove

- D è un dominio non vuoto
- $\{R_1, \ldots, R_m\}$ assegna una relazione n-aria R_i a ciascun predicato n-ario p_i
- $\{d_1, \ldots, d_k\}$ assegna un valore $d_j \in D$ a ciascuna costante a_j

Logica del Primo Ordine: Introduzione

La logica proposizionale non permette di operare su insiemi che non siano l'insieme dei valori booleani $\{\top, \bot\}$. Inoltre è un linguaggio di base che non è in grado di esprimere concetti complessi.

Definizione 7: Sia $\mathcal{D}$ un insieme. $\mathcal{R}$ è una *relazione n-aria* sul *dominio* $\mathcal{D}$ se è una relazione su $\mathcal{D}^n$, cioè se $\mathcal{R} \subseteq \underbrace{\mathcal{D} \times \ldots \times \mathcal{D}}_{n}$.

Il predicato R associato ad $\mathcal{R}$ è

$$R(d_1, \ldots, d_n) \text{ iff } (d_1, \ldots, d_n) \in \mathcal{R}$$

L'introduzione dei predicati e dei quantificatori($\forall, \exists$), rende la logica del primo ordine molto più espressiva rispetto alla logica proposizionale. Comunque, quasi tutto quello che è stato visto per la logica proposizionale può essere esteso alla logica del primo ordine.

Definizione 10: A è vera in $\mathcal{I}$ (soddisfacibile) o $\mathcal{I}$ è un modello per A, se $v(A) = \top$ sotto $\mathcal{I}$. Notazione $\mathcal{I} \models A$

Definizione 11: Una formula A è soddisfacibile se per una qualche interpretazione $\mathcal{I}$ (A è vera in $\mathcal{I}$), $\mathcal{I} \models A$. A è valida se per ogni interpretazione $\mathcal{I}$, $\mathcal{I} \models A$

Esempi

$\forall x.A(x) \equiv \neg\exists x.\neg A(x)$

$\exists x.A(x) \equiv \neg\forall x.\neg A(x)$

$\forall x.A(x) \rightarrow \exists x.A(x)$

$(\exists x.A(x) \vee B(x)) \equiv ((\exists x.A(x)) \vee (\exists x.B(x)))$

$(\forall x.A(x) \wedge B(x)) \equiv ((\forall x.A(x)) \wedge (\forall x.B(x)))$

$(\exists x.A(x) \wedge B(x)) \rightarrow ((\exists x.A(x)) \wedge (\exists x.B(x)))$

$((\forall x.A(x)) \vee (\forall x.B(x))) \rightarrow (\forall x.A(x) \vee B(x))$

Logica del Primo Ordine

Definizione 9: Sia A una formula chiusa. $v(A)$, il valore di A sotto una interpretazione $\mathcal{I}$, è ottenuto sostituendo ad ogni costante a_j di A con il corrispondente elemento del dominio d_j di $\mathcal{I}$ e poi per induzione sulla struttura di A:

- se $A = p_i(c_1, \ldots, c_n)$ è una formula atomica, allora $v(A) = \top$ iff $\{c_1, \ldots, c_n\} \in R_i$ (dove R_i è la relazione assegnata da $\mathcal{I}$ a p_i)
- $v(\neg A_1) = \top$ iff $v(A_1) = \bot$
- $v(A_1 \vee A_2) = \top$ iff $v(A_1) = \top$ o $v(A_2) = \top$ e analogamente per gli altri connettivi booleani
- $v(\forall x.A_1) = \top$ iff *per ogni* $c \in D$, $v(A_1[x \leftarrow c]) = \top$
- $v(\exists x.A_1) = \top$ iff *per qualche* $c \in D$, $v(A_1[x \leftarrow c]) = \top$

dove $A[x \leftarrow c]$ = formula A dove x è "sostituito" con c

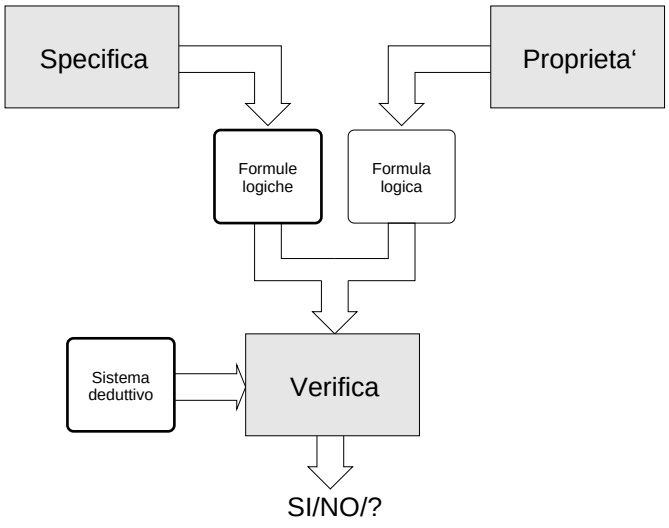
Sistemi deduttivi

Un modo per affrontare la dimostrazione di teoremi è quello di partire da un insieme minimo di assiomi e da un insieme di leggi di deduzione per derivare nuove formule. Tale insieme di assiomi e leggi di deduzione è chiamato *sistema deduttivo* e le nuove formule che vengono dedotte sono chiamate *teoremi*. Il procedimento di deduzione è puramente sintattico e tale procedimento viene chiamato *dimostrazione* del teorema. I vantaggi che questo approccio presenta sono:

- non importa che il numero di assiomi sia finito o infinito
- la dimostrazione fa vedere chiaramente quali assiomi, leggi e teoremi vengono utilizzati e con quale obiettivo
- una volta che un teorema è stato dimostrato, questo può essere utilizzato in ulteriori dimostrazioni come fosse un assioma

Logica per la Specifica

La logica del primo ordine è usata per esprimere vincoli (proprietà) sul sistema che deve essere specificato. La logica è un supporto per *la verifica di proprietà*.



Sistemi deduttivi: completezza e consistenza

Definizione 12: Una dimostrazione in un sistema deduttivo è una sequenza di insiemi di formule tali che ogni elemento è un assioma oppure può essere dedotto da uno o più elementi precedenti con una legge di deduzione. Se A è l'ultimo elemento della sequenza, la sequenza è una dimostrazione di A e A è dimostrabile. Notazione: $\vdash A$.

Definizione 13: Siano U un insieme di formule ed A una formula. la notazione $U \vdash A$ indica che le formule in U sono *assunzioni* nella dimostrazione di A e quindi possono essere utilizzate nella dimostrazione come se fossero assiomi.

Definizione 14: Un sistema deduttivo si dice *completo* (*complete*) se è in grado di costruire una dimostrazione per ogni formula valida, si dice *consistente* (*sound*) se ogni formula per cui è possibile costruire una dimostrazione è effettivamente valida.

Sistemi deduttivi

Il problema che questo nuovo approccio introduce è che, essendo un modo di effettuare le dimostrazioni più vicino a quello manuale, non necessita di forza bruta per arrivare al risultato, ma di un opportuno "direzionamento" della dimostrazione.

Esistono due sistemi di assiomi per la logica proposizionale, estendibili aggiungendo opportuni assiomi e leggi di deduzione alla logica del primo ordine: i sistemi di Gentzen e di Hilbert.

Il sistema di Gentzen consiste di una solo assioma e più leggi di deduzione, mentre il sistema di Hilbert comprende più assiomi ed una sola legge di deduzione(nel caso della logica del primo ordine diventano due). Il sistema di Hilbert è una formalizzazione del metodo classico di ragionamento matematico e quindi è più facile da utilizzarsi per dimostrare teoremi anche manualmente. Noi esamineremo soltanto il sistema di Hilbert perché appare più naturale.

Sistemi deduttivi : leggi derivate

Esempi di leggi derivate:

Legge di deduzione $\dfrac{U \cup \{A\} \vdash B}{U \vdash A \to B}$

Legge di contrapposizione $\dfrac{\vdash \neg B \to \neg A}{\vdash A \to B}$

Legge di transitività $\dfrac{U \vdash A \to B \quad U \vdash B \to C}{U \vdash A \to C}$

Scambio dell'antecedente $\dfrac{U \vdash A \to (B \to C)}{U \vdash B \to (A \to C)}$

Doppia negazione $\dfrac{\vdash \neg\neg A}{\vdash A}$

Sistemi deduttivi : Sistema di Hilbert

Sistema di Hilbert per la logica del primo ordine (consistente e completo):

assioma 1 $\vdash (A \to (B \to A))$

assioma 2 $\vdash (A \to (B \to C)) \to ((A \to B) \to (A \to C))$

assioma 3 $\vdash (\neg B \to \neg A) \to (A \to B)$

assioma 4 $\vdash (\forall x.A(x)) \to A(a)$ if a is a free term for x

assioma 5 $\vdash (\forall x.A \to B(x)) \to (A \to \forall x.B(x))$ A does not depend on x

modus ponens $\dfrac{\vdash A \quad \vdash A \to B}{\vdash B}$

generalizzazione $\dfrac{\vdash A(a)}{\vdash \forall x.A(x)}$

Sistemi deduttivi : esempio(2)

$$\vdash (A \to B) \to ((B \to C) \to (A \to C))$$

1.	$\{A \to B, B \to C, A\} \vdash A$	premessa
2.	$\{A \to B, B \to C, A\} \vdash A \to B$	premessa
3.	$\{A \to B, B \to C, A\} \vdash B$	MP 1,2
4.	$\{A \to B, B \to C, A\} \vdash B \to C$	premessa
5.	$\{A \to B, B \to C, A\} \vdash C$	MP 3,4
6.	$\{A \to B, B \to C\} \vdash A \to C$	deduzione 5
7.	$\{A \to B\} \vdash (B \to C) \to (A \to C)$	deduzione 6
8.	$\vdash (A \to B) \to ((B \to C) \to (A \to C))$	deduzione 7

Sistemi deduttivi : esempio(1)

Legge di contrapposizione

$$\frac{\vdash \neg B \to \neg A}{A \to B}$$

1.	$\vdash \neg B \to \neg A$	premessa
2.	$\vdash (\neg B \to \neg A) \to (A \to B)$	assioma 3
3.	$A \to B$	MP 1,2

Dimostratori automatici di teoremi

Considerando appunto la non decidibilità nel caso generale del problema della soddisfacibilità e della validità di formule della logica del primo ordine, i tentativi di affrontare la dimostrazione di teoremi in modo "assistito" dal calcolatore si possono suddividere in due categorie:

- dimostratori completamente automatici che provano ad applicare varie procedure per tentare di dimostrare un teorema con minimo intervento esterno (solitamente questo si riduce ad indicare gli assiomi della teoria e nell'aggiustare alcuni parametri per velocizzare la dimostrazione o provare a dirigerla in una certa direzione)
- dimostratori semiautomatici che mettono a disposizione strumenti automatici (per svolgere i compiti più elementari, ripetitivi e per dimostrare lemmi più semplici necessari per la dimostrazione del teorema), ma che richiedono un continuo intervento umano per effettuare scelte al livello più alto (praticamente sono degli *assistenti dimostratori* che assicurano la correttezza dei passaggi e svolgono compiti più semplici, al massimo evidenziando i prossimi passi possibili)

Indecidibilità e complessità

Come si è finora visto, la logica del primo ordine è "trattabile", cioè esistono sistemi deduttivi consistenti e completi (Hilbert e Gentzen) e "procedure di decisione" (tabelle semantiche).

Purtroppo vi sono due problemi che si presentano:

- la logica del primo ordine è **indecidibile**, ossia non esiste un algoritmo generale in grado di valutare la soddisfacibilità o la validità di formule generiche
- vi sono classi di formule della logica del primo ordine per le quali una procedura di decisione esiste, ma anche per tali classi spesso la **complessità** per valutare la soddisfacibilità o la validità di una formula è troppo elevata per essere affrontata con gli strumenti di calcolo che si hanno a disposizione

Leggi di riscrittura(1)

Sono metodi basati sul concetto di uguaglianza. Utilizza equazioni come leggi di riscrittura per verificare una formula è una consequenza di un dato insieme di formule.

Osservazione 1: L'insieme di leggi di riscrittura deve avere alcune proprietà:

- non deve essere possibile applicare indefinitamente le leggi di riscrittura (ad esempio una legge del tipo $a \rightarrow f(a)$ è inadeguata) altrimenti il procedimento non ha termine
- non deve accadere che il risultato finale dipenda dall'ordine di applicazione delle leggi quando più leggi sono applicabili nello stesso momento

Esiste una procedura (*Knuth-Bendix completion procedure*) per l'applicazione di leggi di riscrittura che è stata utilizzata per costruire vari sistemi di riscrittura per dimostrazione di teoremi (spesso in connessione a definizioni di ADT)

Tecniche per la dimostrazione automatica

Le principali tecniche disponibili per essere utilizzate per la dimostrazione "assistita" di teoremi sono:

- Tabelle semantiche
- Leggi di riscrittura
- Risoluzione
- Sistemi basati sui "sequent" (*sequent calculus*)
- Sistemi deduttivi

Mentre *tabelle semantiche* e *leggi di riscrittura* sono tecniche fondamentalmente automatiche, la *risoluzione* e il *sequent calculus* si prestano ad essere utilizzate sia in modo automatico che semiautomatico, i *sistemi deduttivi* infine si prestano maggiormente ad un utilizzo interattivo in *assistenti dimostratori*.

Leggi di riscrittura(3)

I principali problemi con le leggi di riscrittura sono:

- il procedimento non funziona sempre: talvolta applicando leggi di induzione alla variabile sbagliata entrano in "loop" (anche se applicare l'induzione ad un'altra variabile porterebbe ad una dimostrazione semplice)
- l'applicazione delle leggi di riscrittura è fatta in modo automatico e quindi quando il sistema entra in loop l'uomo non può intervenire a risolvere il problema direzionando la riscrittura (d'altra parte visto l'elevato numero di riscritture necessarie per dimostrare un teorema l'automatismo diventa necessario)

Leggi di riscrittura(2)

Esempio: $x + s(y) = s(x + y)$ può essere orientata come la legge di riscrittura $x + s(y) \rightarrow s(x + y)$ ed insieme a $x + 0 \rightarrow x$ venire utilizzata per provare che $(x + 0) + s(x + y) = x + (x + s(y))$ provando così il teorema:

$$\{x + 0 = x, x + s(y) = s(x + y)\} \vdash (x + 0) + s(x + y) = x + (x + s(y))$$

Infatti:

1. $(x + 0) + s(x + y) = x + (x + s(y))$
2. $(x) + s(x + y) = x + (x + s(y))$
3. $x + s(x + y) = x + (x + s(y))$
4. $s(x + (x + y)) = x + (x + s(y))$
5. $s(x + (x + y)) = x + s(x + y)$
6. $s(x + (x + y)) = s(x + (x + y))$

Deduzione naturale: sistema deduttivo

Assioma di assunzione

$$\frac{}{\Gamma, \alpha \vdash \alpha}$$

Leggi di introduzione di assunzione

$$\frac{\Gamma \vdash \beta}{\Gamma, \alpha \vdash \beta}$$

Deduzione naturale

Utilizzare praticamente il sistema deduttivo di Hilbert porta a dimostrazioni lunghe e tediose. Tale sistema non è stato pensato per essere il supporto per la dimostrazione, ma per fornire una base che permetta di dimostrare la *soundness* e la *completeness* di un sistema di assiomi per la logica del primo ordine.

Tale sistema però ci consente di costruire un insieme di leggi derivate e svolgere le dimostrzioni utilizzando tali leggi. Le leggi per la *deduzione naturale* sono state pensate per permettere un modo di ragionare più vicino al ragionamento matematico classico nello svolgere le dimostrazioni.

La deduzione naturale permette due diversi approcci alla dimostrazione: diretto (dagli assiomi all'obiettivo desiderato, cioè il teorema) e inverso (a partire dall'obiettivo, cioè il teorema, desiderato passando attraverso dei sottoobiettivi per risalire fino agli assiomi).

Deduzione naturale: Leggi di introduzione ed eliminazione(2)

	Introduzione	Eliminazione
$\forall$	$\dfrac{\Gamma\vdash\alpha(c)}{\Gamma\vdash\forall x.\alpha(x)}$	$\dfrac{\Gamma\vdash\forall x.\alpha(x)}{\Gamma\vdash\alpha(t)}$
	c non presente in nessun altro membro di Γ	t è variab. libera per x in $\alpha(x)$
$\exists$	$\dfrac{\Gamma\vdash\alpha(t)}{\Gamma\vdash\exists x.\alpha(x)}$	$\dfrac{\Gamma\vdash\exists x.\alpha(x)\quad\Gamma,\alpha(c)\vdash\beta}{\Gamma\vdash\beta}$
	dove t è variab. libera per x in $\alpha(x)$	dove c non presente in Γ,$\exists x.\alpha(x)$, β
$=$	$\dfrac{}{\Gamma\vdash t=t}$	$\dfrac{\Gamma\vdash t_1=t_2}{\Gamma\vdash\alpha(t_1)=\alpha(t_2)}$
		dove t_1, t_2 sono variabili libere per x in $\alpha(x)$

Deduzione naturale: Leggi di introduzione ed eliminazione(1)

	Introduzione	Eliminazione
$\neg$	$\dfrac{\Gamma,\alpha\vdash\beta\quad\Gamma,\alpha\vdash\neg\beta}{\Gamma\vdash\neg\alpha}$	$\dfrac{\Gamma\vdash\alpha\quad\Gamma\vdash\neg\alpha}{\Gamma\vdash\beta}$
$\neg\neg$	$\dfrac{\Gamma\vdash\alpha}{\Gamma\vdash\neg\neg\alpha}$	$\dfrac{\Gamma\vdash\neg\neg\alpha}{\Gamma\vdash\alpha}$
$\vee$	$\dfrac{\Gamma\vdash\gamma}{\Gamma\vdash\alpha\vee\beta}$	$\dfrac{\Gamma,\alpha\vdash\gamma\quad\Gamma,\beta\vdash\gamma\quad\Gamma\vdash\alpha\vee\beta}{\Gamma\vdash\gamma}$
	dove γ è α o β	
$\wedge$	$\dfrac{\Gamma\vdash\alpha\quad\Gamma\vdash\beta}{\Gamma\vdash\alpha\wedge\beta}$	$\dfrac{\Gamma\vdash\alpha\wedge\beta}{\Gamma\vdash\gamma}$
		dove γ è α o β
$\rightarrow$	$\dfrac{\Gamma,\alpha\vdash\beta}{\Gamma\vdash\alpha\rightarrow\beta}$	$\dfrac{\Gamma\vdash\alpha\quad\Gamma\vdash\alpha\rightarrow\beta}{\Gamma\vdash\gamma}$

Isabelle: formalizzazione di logiche

Per formalizzare una logica occorre definirne la sintassi e la semantica.

Per la definizione della sintassi occorre definire quali sono i simboli del linguaggio (operatori) e qual è la loro *segnatura*, cioè il dominio dei loro operandi e del loro risultato.

Per la definizione della semantica di un operatore ci sono due strumenti:

- leggi deduttive di introduzione ed eliminazione per l'operatore
- definizione dell'operatore in base ad altri operatori già definiti

Il secondo metodo (non sempre possibile) è più sicuro perché Isabelle verifica che le definizioni siano corrette (cioè siano davvero definizioni) ed è quindi preferibile. Il primo metodo invece permette l'introduzione di inconsistenze (p.e.: $\top \Rightarrow \bot$).

Isabelle

Isabelle è un dimostratore generico di teoremi. È generico nel senso che non è legato ad una logica particolare (logica proposizionale, logica primo ordine o altro), ma mette a disposizione una meta-logica di *ordine superiore* tramite la quale è possibile descrivere ogni altro tipo di logica.

Il sistema mette a disposizione una serie di logiche già formalizzate per le quali sono disponibili strumenti per la dimostrazione automatica e numerosi teoremi predimostrati.

La formalizzazione di logiche avviene attraverso la specifica di un insieme di assiomi (leggi di deduzione e definizioni di operatori) che definiscono quale è la sintassi della logica e quale è la sua semantica. La semantica viene data in modo assiomatico definendo un modo per assegnare un valore di verità ad una formula.

Per ogni logica si possono poi dimostrare teoremi e costruire strumenti per la dimostrazione automatica utilizzando i teoremi via via dimostrati ed un opportuno linguaggio di programmazione.

Isabelle

Isabelle distingue i termini in 3 categorie:

- variabili libere: variabili che vengono introdotte fin dall'inizio nel teorema senza essere quantificate e si considerano implicitamente quantificate universalmente
- parametri: vengono introdotti durante la dimostrazione e provengono da variabili quantificate
- incognite: vengono introdotte durante la dimostrazione e possono prendere un valore (dipendente dalle variabili libere e da parte dei parametri) per provare dei sotto-obiettivi del teorema

Isabelle: dimostrazione di teoremi

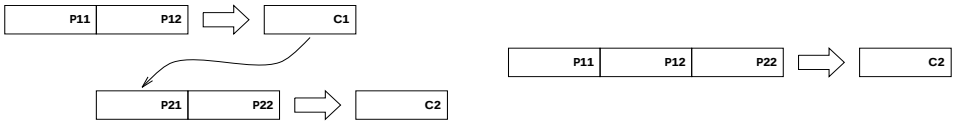
Sono possibili due metodi di ragionamento:

- dimostrazioni *costruttive* a partire da assiomi e leggi di deduzione
- dimostrazioni *all'indietro*, cioè a partire dal teorema che si vuole dimostrare risalendo fino a fatti elementari ottenuti scomponendo il teorema tramite l'applicazione *alla rovescia* di leggi di deduzione.

La dimostrazione costruttiva è possibile solo in modo manuale, mentre per quelle all'indietro (*dimostrazioni dirette dagli obiettivi*) sono disponibili numerosi strumenti per procedere in modo automatico:

- insiemi di leggi di riscrittura
- insiemi di leggi di deduzione (per procedimento deduttivo all'indietro)
- linguaggio per la combinazione di passi base per costruire strategie di ricerca della soluzione *ad hoc*

Isabelle: dimostrazioni in avanti



La dimostrazione in avanti consiste nel combinare insieme teoremi per crearne di nuovi. Tramite una funzione viene effettuata l'unificazione tra la conclusione di un teorema e la premessa di un altro (es.: C1 e P11). Il teorema risultante ha, come conclusione, la conclusione del secondo teorema e, come premesse ,le premesse dei due teoremi meno la premessa del secondo che è stata utilizzata per l'unificazione. Esempio:

sym	$s = t \Rightarrow t = s$
trans	$[\![r = s;\ \ s = t]\!] \Rightarrow r = t$
sym RS trans	$[\![s = r;\ \ s = t]\!] \Rightarrow r = t$
trans RS sym	$[\![s = s1;\ \ s1 = t]\!] \Rightarrow t = s$

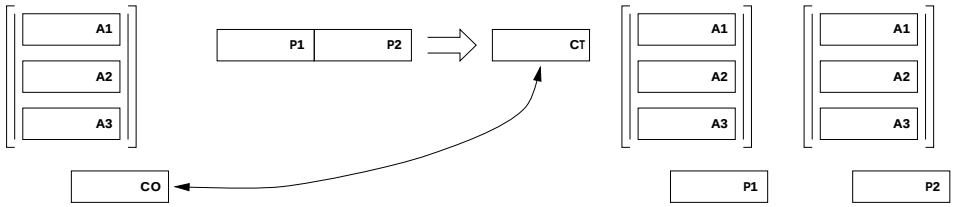
Isabelle

I teoremi in Isabelle vengono rappresentati da una serie di premesse ed una conclusione:

$$\frac{P_1; \ldots; P_n}{Q}$$

tale forma viene anche utilizzata per rappresentare lo stato di una dimostrazione nel caso di dimostrazioni guidate dall'obiettivo ($P_1; \ldots; P_n$ sono i sotto-obiettivi che occorre provare per dimostrare il teorema Q).

Isabelle: risoluzione



Esempio

Goal:

$[\![A; B; C]\!] \Longrightarrow (A \vee B) \wedge C$

Si usa il teorema:

$[\![X; Y]\!] \Longrightarrow X \wedge Y \qquad [X \leftarrow (A \vee B); Y \leftarrow C]$

Si ottengono due sub-goal:

$[\![A; B; C]\!] \Longrightarrow (A \vee B)$

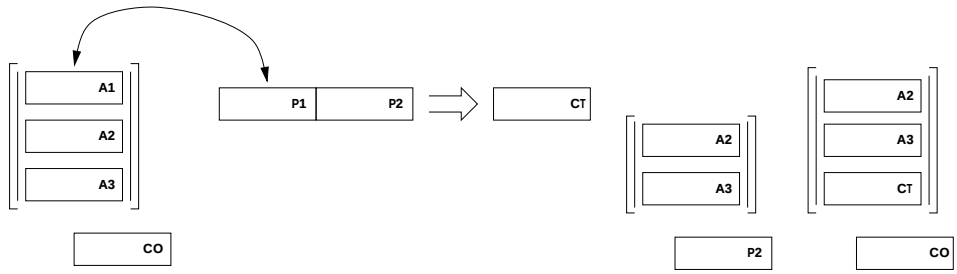
$[\![A; B; C]\!] \Longrightarrow C$

Isabelle: dimostrazioni all'indietro

Ci sono tre metodi per procedere nelle dimostrazioni all'indietro:

- **risoluzione** unificando la conclusione, CO, di un sotto-obiettivo con la conclusione di un teorema, CT; si genera un nuovo sotto-obiettivo per ogni premessa del teorema applicato, nel quale tale premessa diventa la conclusione da dimostrare e le assunzioni del sotto-obiettivo originale diventano le assunzioni del nuovo sotto-obiettivo
- **risoluzione con eliminazione**, variazione nella quale viene unificata anche una assunzione del sotto-obiettivo con la prima premessa del teorema ($A1 = P1$) (N.B.: l'assunzione unificata viene eliminata e le altre premesse del teorema vanno a far parte delle assunzioni del sotto-obiettivo)
- **risoluzione con distruzione**, permette il ragionamento in avanti dalle assunzioni, effettuando l'unificazione tra un'assunzione del sotto-obiettivo (che viene eliminata) e la prima premessa del teorema applicato; si ottiene un nuovo sotto-obiettivo analogo all'originario solo che la premessa eliminata è sostituita dalla conclusione del teorema. Inoltre sono aggiunti dei sotto-obiettivi con le premesse originali (senza quella eliminata) e con conclusione ognuno con una premessa del teorema.

Isabelle: risoluzione con distruzione



Esempio

Goal:

$[\![A \wedge B \to C; A; B]\!] \Longrightarrow C$

Si usa il teorema:

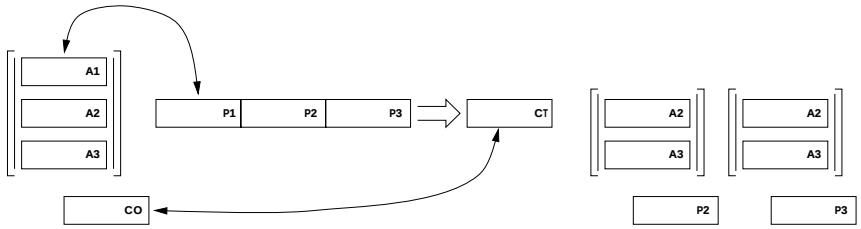
$[\![X \to Y; X]\!] \Longrightarrow Y \qquad [X \leftarrow A \wedge B; Y \leftarrow C]$

Si ottengono due sub-goal:

$[\![A; B]\!] \Longrightarrow A \wedge B$

$[\![A; B; C]\!] \Longrightarrow C$

Isabelle: risoluzione con eliminazione



Esempio

Goal:

$[\![A \vee B; A \to C; B \to C]\!] \Longrightarrow C$

Si usa il teorema:

$[\![X \vee Y; X \Longrightarrow R; Y \Longrightarrow R]\!] \Longrightarrow R \qquad [X \leftarrow A; Y \leftarrow B; R \leftarrow C]$

Si ottengono due sub-goal:

$[\![A \to C; B \to C; A]\!] \Longrightarrow C$

$[\![A \to C; B \to C; B]\!] \Longrightarrow C$

Logiche Temporali

Le logiche temporali si distinguono dalle logiche che abbiamo visto finora per il fatto che sono pensate per ragionare su proposizioni il cui valore di verità varia nel *tempo*. Sono molto interessanti in informatica perché sia l'hardware che il software impiegano componenti il cui stato cambia nel tempo.

Il tempo solitamente viene rappresentato da una successione di *mondi* e ciascuna espressione logica può avere un diverso valore logico in ogni istante (*mondo*).

Esempi di comportamenti che si possono descrivere con le logiche temporali comprendono:

- Se è stata fatta una richiesta di stampare un file, *prima o poi* (*eventually*) il file verrà stampato.
- Quando il loop while terminerà il valore della variabile `s` sarà uguale alla somma degli elementi dell'array.
- Se il mouse viene clickato sul bottone "Browse", la finestra di selezione di un file deve stare aperta finché un file è stato selezionato

Isabelle: strumenti automatici

In Isabelle sono presenti due strumenti di supporto alla dimostrazione automatica:

- il *Semplificatore* che sfruttando un insieme di teoremi del tipo $A = B$ oppure $C \Rightarrow A = B$ semplifica un goal.
- il *Classical Reasoner* che applica teoremi di introduzione, eliminazione o distruzione in modo automatico (teoremi safe e unsafe).

Logiche Temporali: tempo lineare

Le logiche temporali sono divise in due famiglie: *tempo lineare* e *tempo ramificato*. Si hanno logiche con tempo lineare se per ogni stato si ammette un solo possibile stato futuro, altrimenti si hanno logiche con tempo ramificato se in ogni stato possono esserci più stati futuri. Questo ultimo tipo di logiche si possono usare per modellare programmi non deterministici.

Ultima limitazione che introduciamo è che il tempo sia discreto (cosa vera per la maggior parte dei casi di interesse, quali la descrizione di esecuzioni di programmi). Si può quindi introdurre anche l'operatore $\bigcirc$, con $\bigcirc p$ che indica che p sarà vera nel prossimo stato. Per logiche a tempo discreto valgono:

$$\models \Box A \rightarrow \bigcirc A$$
$$\models \bigcirc A \rightarrow \Diamond A$$
$$\models \bigcirc A \leftrightarrow \neg \bigcirc \neg A$$

Logiche Temporali: sintassi

La logica temporale proposizionale è costituita dalla logica proposizionale con aggiunti due operatori temporali unari:

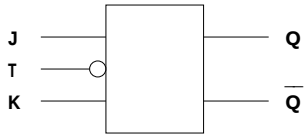
- operatore $\Box$ che si legge "*always*" o "*henceforth*"
- operatore $\Diamond$ che si legge "*eventually*" o "*sometime*"

L'operatore $\Box$ significa che la formula logica alla quale è applicato è vera nello stato attuale e sarà vera in ogni stato successivo. L'operatore $\Diamond$ significa che la formula logica alla quale è applicato è vera nello stato attuale oppure sarà vera in uno stato futuro.

$$\Box P \equiv P_0 \wedge \forall s.s > 0 \rightarrow P_s$$
$$\Diamond P \equiv P_0 \vee \exists s.s > 0 \rightarrow P_s$$

Logiche Temporali: esempio(1)

Consideriamo un flip-flop di tipo JK



Un cambiamento di stato nel flip-flop avviene sui fronti di discesa di T. Tale evento verrà indicato con $T\downarrow$ Gli stati ammissibili possono essere caratterizzati dalle seguenti formule:

$$(T\downarrow \wedge (J=0)\wedge(K=1)) \rightarrow \bigcirc(Q=0)$$
$$(T\downarrow \wedge (J=1)\wedge(K=0)) \rightarrow \bigcirc(Q=1)$$
$$(T\downarrow \wedge (J=1)\wedge(K=1)) \rightarrow (Q=v \rightarrow \bigcirc(Q=1-v))$$
$$(T\downarrow \wedge (J=0)\wedge(K=0)) \rightarrow (Q=v \rightarrow \bigcirc(Q=v))$$
$$\Box(Q=v \leftrightarrow \overline{Q}=1-v)$$

Logiche Temporali: sistema deduttivo

Un sistema deduttivo (estendendo il sistema di Hilbert) per la logica PTL si ottiene aggiungendo ai tre assiomi della logica temporale ed al *modus ponens* i seguenti assiomi:

assioma 4 $\vdash \Box(A\rightarrow B)\rightarrow(\Box A\rightarrow\Box B)$

assioma 5 $\vdash \bigcirc(A\rightarrow B)\rightarrow(\bigcirc A\rightarrow\bigcirc B)$

assioma 6 $\vdash \Box A\rightarrow(A\wedge\bigcirc A\wedge\bigcirc\Box A)$

assioma 7 $\vdash \Box(A\rightarrow\bigcirc A)\rightarrow(A\rightarrow\Box A)$

assioma 8 $\vdash \bigcirc A\leftrightarrow\neg\bigcirc\neg A$

e la legge di generalizzazione:

$$\frac{\vdash A}{\vdash \Box A}$$

Logiche Temporali: estensioni(1)

Definizione 15: $s_0 \models p\mathcal{W}q$ se

- per ogni i, $s_i \models p$ oppure
- esiste n tale che $s_n \models q$ e per ogni i tale che $0 \leq i < n$, $s_i \models p$.

L'operatore $\mathcal{W}$ si chiama *waiting* e $p\mathcal{W}q$ è vero se p rimane vero finché q diventa vero.

Ora l'ultima specifica per il flip-flop può essere espressa con:

$$((Q = v) \rightarrow \bigcirc(Q = v))\mathcal{W}(T\downarrow)$$

Logiche Temporali: esempio(2)

La prima legge descrive il reset del flip-flop: quando ho un fronte di discesa ($T\downarrow$) e $J = 0$ e $K = 1$ allora al prossimo stato avrò $Q = 0$.

La seconda descrive analogamente il set del flip-flop: quando ho un fronte di discesa ($T\downarrow$) e $J = 1$ e $K = 0$ allora al prossimo stato avrò $Q = 1$.

La terza legge descrive la possibilità di complementare il flip-flop.

La quarta legge descrive la possibilità di lasciare invariato lo stato del flip-flop.

La quinta asserisce infine che il valore di $\overline{Q}$ è sempre il complemento di Q.

Osservazione 2: La specifica finora data non è completa. Infatti, occorre specificare che i valori di Q e $\overline{Q}$ non variano fino al prossimo fronte di discesa di T. Il problema è che, con gli operatori fin qui definiti, non è possibile esprimere questo concetto e quindi occorrerà aggiungere altri operatori per estendere l'universo delle formule della logica temporale proposizionale.

Logiche Temporali: estensioni(3)

Ciascuno degli operatori binari $\mathcal{W}$ e $\mathcal{U}$ è sufficiente a descrivere tutti gli altri operatori (tranne il $\bigcirc$), infatti:

$$\begin{aligned} \models \Diamond p \quad &\leftrightarrow \quad \top \mathcal{U} p \\ \models \Box p \quad &\leftrightarrow \quad p \mathcal{W} \bot \end{aligned}$$

Gli assiomi da aggiungere nel sistema deduttivo per poter trattare questi nuovi operatori sono:

assioma 9 $\vdash p\mathcal{W}q \leftrightarrow (q \vee (p \wedge \bigcirc(p\mathcal{W}q)))$

assioma 10 $\vdash \Box p \rightarrow p\mathcal{W}q$

Logiche Temporali: estensioni(2)

Esiste una versione più forte dell'operatore $\mathcal{W}$, $p\mathcal{U}p$ che assicura che prima o poi q diventerà vera.

Definizione 16: $s_0 \models p\mathcal{U}q$ se

- esiste n tale che $s_n \models q$ e
- per ogni i tale che $0 \leq i < n$, $s_i \models p$.

Teorema 2:

$$\begin{aligned} \models p\mathcal{U}q \quad &\leftrightarrow \quad (p\mathcal{W}q) \wedge \Diamond q \\ \models p\mathcal{W}q \quad &\leftrightarrow \quad (p\mathcal{U}q) \vee \Box p \\ \models \neg(p\mathcal{U}q) \quad &\leftrightarrow \quad (\neg q)\mathcal{W}(\neg p \wedge \neg q) \\ \models \neg(p\mathcal{W}q) \quad &\leftrightarrow \quad (\neg q)\mathcal{U}(\neg p \wedge \neg q) \end{aligned}$$

Sistemi tempo-reale

Vediamo alcuni esempi:

- A deve essere seguito da B entro 3 unità di tempo
- A deve essere seguito da B in 7 unità di tempo esatte
- due A consecutivi sono distanziati da almeno 5 unità di tempo
- l'evento E accade ogni 4 unità di tempo
- ad ogni occorrenza di E devo rispondere con D entro 3 unità di tempo

Sistemi tempo-reale

La differenza tra un sistema di tipo tradizionale ed un sistema tempo-reale sta nel fatto che per quest'ultimo i vincoli temporali costituiscono la definizione di comportamento corretto e non caratterizzano soltanto la bontà delle prestazioni del sistema.

In altre parole, mentre per un sistema normale i vincoli temporali possono solo descrivere delle richieste sulla "velocità" o "protezza" di risposta del sistema (che entro certi limiti possono non essere soddisfatte), per un sistema tempo-reale i vincoli temporali definiscono quali comportamenti sono corretti e quali sono equivalenti al verificarsi di condizioni di errore.

Sistemi tempo-reale

Ad esempio, la formula temporale $\Box(A \rightarrow \Diamond B)$ dice che dopo che l'evento A è accaduto accadrà l'evento B, ma non è possibile porre alcun vincolo sulla distanza tra i due eventi (al massimo si potrà imporre che B avvenga prima di qualche altro evento).

Ciò che quindi manca è una *metrica* del tempo, cioè la possibilità di misurare la distanza fra eventi.

È invece possibile specificare tutte le possibili relazioni di precedenza e di ordinamento tra eventi lasciando implicite le distanze tra tali eventi.

Sono state sviluppate alcune logiche temporali con metrica del tempo che quindi possono essere impiegate nello sviluppo di sistemi tempo-reale.

Tutte queste logiche temporali si basano sulla introduzione di una metrica del tempo che sia in grado di misurare la distanza temporale tra eventi e conseguentemente permetta di esprimere vincoli su tale distanza.

Sistemi tempo-reale

Una classificazione dei tipi di requisiti che possono essere necessari nella descrizione di sistemi tempo-reale può essere la seguente:

1. **Tempo di risposta**: mette in relazione un evento e la reazione a tale evento
   (a) massima distanza tra evento e reazione (*time-out*)
   (b) distanza esatta tra evento e reazione (*delay*)
2. **Frequenza**: mette in relazione due occorrenze dello stesso evento
   (a) minima distanza tra due occorrenze
   (b) periodicità o distanza esatta tra occorrenze

Chiaramente mentre le logiche temporali tradizionali possono essere sufficienti a descrivere le caratteristiche di sistemi tradizionali (anche quando vi sia necessità di descrivere sottosistemi in esecuzione concorrente tra loro), tale supporto non è più sufficiente nel caso di sistemi tempo-reale. Infatti, per questi appare fondamentale l'espressione di vincoli temporali in forma quantitativa.

TILCO: esempi

Riprendiamo alcuni esempi:

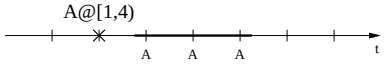
- A deve essere seguito da B entro 3 unità di tempo
  $$A \rightarrow \neg B \wedge (B?(0,3])$$
- A deve essere seguito da B in 7 unità di tempo esatte
  $$A \rightarrow \neg B@[0,7) \wedge B@[7,7]$$
- due A consecutivi sono distanziati da almeno 5 unità di tempo
  $$A \rightarrow \neg A@(0,5)$$
- l'evento E accade ogni 4 unità di tempo
  $$E \rightarrow \neg E@(0,4) \wedge E@[4,4]$$
  $$E?(-\infty,+\infty)$$
- ad ogni occorrenza di E devo rispondere con D entro 3 unità di tempo
  $$E \rightarrow D?(0,3]$$

TILCO

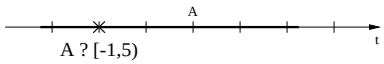
TILCO = Temporal Interval Logic with Compositional Operators

- Logica temporale del primo ordine ($\wedge$,$\vee$,$\neg$,$\rightarrow$,$\forall$, $\exists$);
- Tempo discreto;
- Tempo lineare;
- Operatori temporali di base:

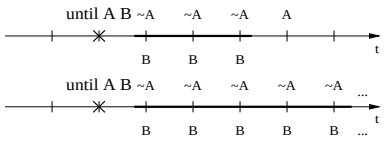
at A @ interval



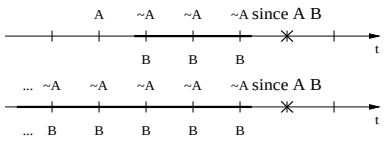
happen A ? interval



until until A B



since since A B



TILCO: esempio(1)

Consideriamo il sistema di atterraggio per un astronave per l'esplorazione di Marte. Il sistema opera in due modalità: normale o emergenza. Quando viene acceso il sistema opera in modalità normale. In modalità normale il pilota controlla accelerazione, velocità e posizione del veicolo con la spinta verso il basso generata da un motore a razzo, portando il veicolo ad un atterraggio sicuro. Il controllo dell'atterraggio avviene in due fasi che si susseguono continuamente. Nella prima fase viene letto il valore di accelerazione impostato dal pilota e viene settato un timer che genera una interrupt dopo 100 ms. Nella seconda fase (quando la lettura è stata completata) la potenza del razzo è opportunamente regolata. Comunque, se l'operazione di lettura non viene completata entro i 100 ms, l'interrupt fa passare in modalità di atterraggio di emergenza. In modalità di emergenza, altitudine e velocità del veicolo vengono letti periodicamente ed un retro-razzo viene automaticamente acceso per far atterrare il veicolo in modo sicuro.

TILCO: operatori derivati

next	$A \twoheadrightarrow B \equiv A \rightarrow (B \ @ \ [1,1])$
prev	$A \twoheadleftarrow B \equiv A \rightarrow (B \ @ \ [-1,-1])$
until'	until' A B $\equiv$ A ? $(0, +\infty)\wedge$ until A B
since'	since' A B $\equiv$ A ? $(-\infty, 0)\wedge$ since A B
until0	until_0 A B $\equiv$ A $\vee$ (B $\wedge$ until A B) $\equiv$ (until A B)@[-1,-1]
since0	since_0 A B $\equiv$ A $\vee$ (B $\wedge$ since A B)
until0'	until_0' A B $\equiv$ A ? $[0, +\infty)\wedge$ until_0 A B
since0'	since_0' A B $\equiv$ A ? $(-\infty, 0]\wedge$ since_0 A B
tuntil	tuntil t A B $\equiv$ (A ? $(0,t]\wedge$ until A B) $\vee$ B @ $(0,t]$
tuntil'	tuntil' t A B $\equiv$ (A ? $(0,t]\wedge$ until A B)
tuntil0	tuntil_0 t A B $\equiv$ (A ? $[0,t]\wedge$ until_0 A B) $\vee$ B @ $[0,t]$
tuntil0'	tuntil_0' t A B $\equiv$ (A ? $[0,t]\wedge$ until_0 A B)

$\mathrm{rule}(\mathrm{InputData} \wedge \neg \mathrm{InputData}@[-1,-1] \rightarrow (\neg(\mathrm{InputData} \wedge \neg \mathrm{InputData}@[-1,-1])@(0,200)))$

$\mathrm{rule}(\mathrm{CheckData} \rightarrow (\neg \mathrm{CheckData}?(0,100)))$

$\mathrm{rule}(\mathrm{CheckData} \wedge \neg \mathrm{CheckData}@[-1,-1] \rightarrow (\neg(\mathrm{CheckData} \wedge \neg \mathrm{CheckData}@[-1,-1])@(0,200)))$

$\mathrm{rule}(\mathrm{RetroRazzo} \rightarrow (\neg \mathrm{RetroRazzo}?(0,100)))$

$\mathrm{rule}(\mathrm{RetroRazzo} \wedge \neg \mathrm{RetroRazzo}@[-1,-1] \rightarrow$

$(\neg(\mathrm{RetroRazzo} \wedge \neg \mathrm{RetroRazzo}@[-1,-1])@(0,200)))$

$\mathrm{rule}(\mathrm{ReadAcc} \rightarrow (\mathrm{until}(\mathrm{IOInt})(\neg \mathrm{Done})))$

$\mathrm{rule}(\mathrm{IOInt} \rightarrow (\mathrm{until}(\mathrm{ReadAcc})(\mathrm{Done})))$

$\mathrm{rule}(\mathrm{IOInt} \rightarrow ((\mathrm{AdjMotor} \wedge \mathrm{AdjDisplay})?(0,60)))$

$\mathrm{rule}(\mathrm{IOInt} \rightarrow (\neg \mathrm{IOInt}@(0,100)))$

$\mathrm{rule}(\mathrm{TimerInt} \wedge \neg \mathrm{Done} \rightarrow ((\mathrm{Stato} = \mathrm{Emergenza})?(0,60)))$

$\mathrm{rule}(\mathrm{TimerInt} \rightarrow (\neg \mathrm{TimerInt}@(0,100)))$

$\mathrm{rule}(\mathrm{StartTimer} \rightarrow (\neg \mathrm{TimerInt}@(0,100)))$

Tilco: esempio(2)

$\mathrm{rule}(A) \equiv \mathrm{now}(A@(-\infty,+\infty))\ 0$

$\mathrm{fact}(A) \equiv \mathrm{now}(A?(-\infty,+\infty))\ 0$

$\mathrm{rule}(\mathrm{Stato} = \mathrm{Off}|\mathrm{Normale}|\mathrm{Emergenza}|\mathrm{Landed})$

$\mathrm{fact}(\mathrm{Stato} = \mathrm{Off}@(-\infty,0))$

$\mathrm{fact}(\mathrm{Done}@(-\infty,0))$

$\mathrm{fact}(\mathrm{Stato} = \mathrm{Normale})$

$\mathrm{fact}(\mathrm{Stato} = \mathrm{Landed}@(0,+\infty))$

$\mathrm{rule}(\mathrm{Stato} = \mathrm{Normale} \rightarrow \mathrm{until}(\mathrm{Stato} = \mathrm{Emergenza} \vee \mathrm{Stato} =$

$\mathrm{Landed})((\mathrm{ReadAcc} \wedge \mathrm{StartTimer})?(0,200)))$

$\mathrm{rule}(\mathrm{ReadAcc} \rightarrow (\neg \mathrm{ReadAcc}?(0,40)))$

$\mathrm{rule}(\mathrm{StartTimer} \rightarrow (\neg \mathrm{StartTimer}?(0,40)))$

$\mathrm{rule}(\mathrm{ReadAcc} \wedge \neg \mathrm{ReadAcc}@[-1,-1] \rightarrow (\neg(\mathrm{ReadAcc} \wedge \neg \mathrm{ReadAcc}@[-1,-1])@(0,200)))$

$\mathrm{rule}(\mathrm{StartTimer} \wedge \neg \mathrm{StartTimer}@[-1,-1] \rightarrow (\neg(\mathrm{StartTimer} \wedge \neg \mathrm{StartTimer}@[-1,-1])@(0,200)))$

$\mathrm{rule}(\mathrm{Stato} = \mathrm{Emergenza} \rightarrow \mathrm{until}(\mathrm{Stato} =$

$\mathrm{Landed})((\mathrm{InputData} \wedge \mathrm{CheckData} \wedge \mathrm{RetroRazzo})?(0,200)))$

$\mathrm{rule}(\mathrm{InputData} \rightarrow (\neg \mathrm{InputData}?(0,100)))$

Tilco: Sistema deduttivo (2)

@I $\dfrac{\dfrac{}{x \in i}}{\mathrm{now}(P)(x+t)}$ (x non variabile libera)

@E $\dfrac{\mathrm{now}(P@i)\ t \quad \dfrac{\mathrm{now}(P)(x+t)}{Q} \quad \dfrac{}{x \notin i}}{Q}$

?I $\dfrac{\mathrm{now}(P)(x+t) \quad x \in i}{\mathrm{now}(P?i)\ t}$

?E $\dfrac{\mathrm{now}(P?i)\ t \quad \dfrac{\mathrm{now}(P)(x+t) \quad x \in i}{R}}{R}$ (x variabile libera solo in P)

untilI1 $\dfrac{\mathrm{now}(P)\ (x+t) \quad 0<x \quad \mathrm{now}(Q@(0,x))\ t}{\mathrm{now}(\mathrm{until}\ P\ Q)t}$

untilI2 $\dfrac{\mathrm{now}(Q@(0,\infty))\ t}{\mathrm{now}(\mathrm{until}\ P\ Q)t}$

untilE $\dfrac{\mathrm{now}(\mathrm{until}\ P\ Q)t \quad \dfrac{\begin{array}{l}\mathrm{now}(P)\ (x+t)\\ 0<x\\ \mathrm{now}(Q@(0,x))\ t\end{array}}{R} \quad \dfrac{\mathrm{now}(Q?(0,+\infty))\ t}{R}}{R}$

Tilco: Sistema deduttivo (1)

$\wedge$E1 $\dfrac{\mathrm{now}(P\wedge Q)\ t \quad \dfrac{\begin{array}{c}\mathrm{now}(P)\ t\\ \mathrm{now}(Q)\ t\end{array}}{R}}{R}$

$\wedge$I $\dfrac{\mathrm{now}(P)\ t \quad \mathrm{now}(Q)\ t}{\mathrm{now}(P\wedge Q)\ t}$

$\vee$E $\dfrac{\mathrm{now}(P\vee Q)\ t \quad \dfrac{\mathrm{now}(P)\ t}{R} \quad \dfrac{\mathrm{now}(Q)\ t}{R}}{R}$

$\vee$I $\dfrac{\dfrac{\neg\mathrm{now}(P)\ t}{\mathrm{now}(Q)\ t}}{\mathrm{now}(P\vee Q)\ t}$

$\rightarrow$I $\dfrac{\begin{array}{c}\mathrm{now}(P)\ t\\ \mathrm{now}(Q)\ t\end{array}}{\mathrm{now}(P\rightarrow Q)\ t}$

MP $\dfrac{\mathrm{now}(P\rightarrow Q)\ t \quad \mathrm{now}(P)\ t}{\mathrm{now}(Q)\ t}$

$\bot$E $\dfrac{\mathrm{now}(\bot)\ t}{P}$

$\top$I $\dfrac{}{\mathrm{now}(\top)\ t}$

$\forall$-def $(\mathrm{now}(\forall x.P(x))\ t) \equiv (\forall x.\mathrm{now}(P(x))\ t)$

$\exists$-def $(\mathrm{now}(\exists x.P(x))\ t) \equiv (\exists x.\mathrm{now}(P(x))\ t)$

$\neg$-def $\mathrm{now}(\neg P)\ t \equiv \neg\mathrm{now}(P)\ t$

$\leftrightarrow$-def $\mathrm{now}(P\leftrightarrow Q)\ t \equiv (\mathrm{now}(P\rightarrow Q)\ t \wedge \mathrm{now}(Q\rightarrow P)\ t)$

Specifica con logiche tradizionali

Le logiche temporali tradizionali non hanno supporto per la strutturazione quindi sono difficilmente utilizzabili per la specifica di sistemi reali.

Queste producono specifiche composte da:

- variabili globali, e
- predicati/formule temporali;

Problemi:

- mancanza di modularizzazione;
- impossibile il riuso di porzioni di specifica;

Tilco: Sistema deduttivo (3)

sinceI1 $$\frac{\text{now}(P)\ (x+t)\quad x<0\quad \text{now}(Q@(x,0))\ t}{\text{now}(\text{since}\ P\ Q)t}$$

sinceI2 $$\frac{\text{now}(Q@(-\infty,0))\ t}{\text{now}(\text{since}\ P\ Q)t}$$

sinceE $$\frac{\text{now}(\text{since}\ P\ Q)t \quad \frac{\text{now}(P)\ (x+t)\quad x<0\quad \text{now}(Q@(x,0))\ t}{R} \quad \frac{\text{now}(Q?(-\infty,0))\ t}{R}}{R}$$

C-Tilco: Processo

Un *processo* è un entità strutturata caratterizzata da:

- variabili;
- predicati;
- porte per la comunicazione con altri processi;
- sotto-processi;
- formule TILCO che specificano il comportamento del processo.

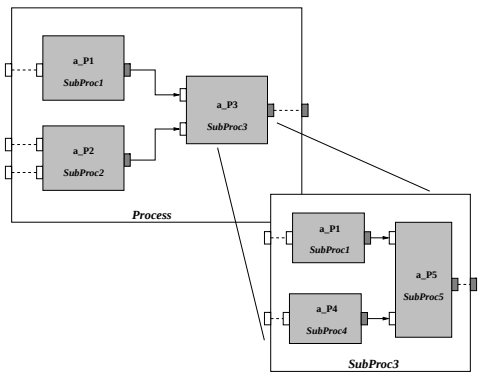
Di un processo possono essere presenti più istanze, ognuna con le proprie variabili, predicati, porte di comunicazione e sotto-processi.
(Processo=Tipo=Entità strutturata)

Notazione: nelle formule i componenti (variabili, predicati, porte e sotto-processi) di un sotto-processo possono essere riferite tramite la dot-notation (es: se s è un sotto-processo che possiede la variabile v allora $s.v$ può essere usato in una formula per riferirsi alla variabile v di s)

C-Tilco

Communicating Tilco

- Sistema come gerarchia di processi comunicanti;
- Metodologia di specifica per decomposizione;
- Base per Esecuzione distribuita

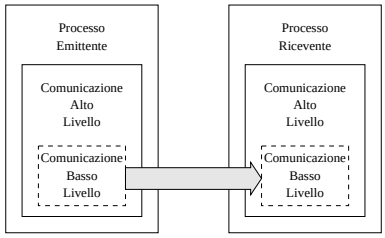


C-Tilco: Comunicazione

La comunicazione ha le seguenti caratteristiche:

- porte di ingresso e di uscita;
- porte tipizzate;
- collegamenti uno a uno, monodirezionali (messaggi);

Comunicazione strutturata su due livelli:



C-Tilco: Specifica processo

Specifica del processo P per raffinamenti successivi:

1. specifica *astratta* del processo P: $\mathcal{S}_a[\![P]\!]$;

1a. verifica e validazione specifica astratta;

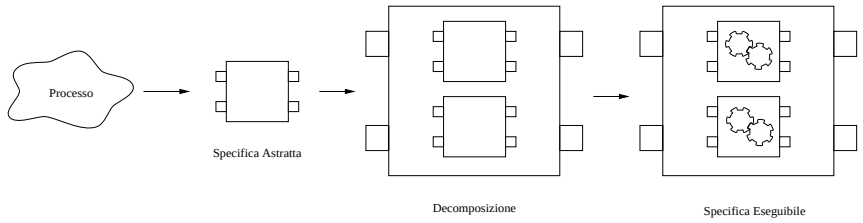
2. eventuale decomposizione di P nei sotto-processi $P_1 \ldots P_n$;

2a. verifica decomposizione:

$$\mathcal{S}_a[\![P_1]\!] \wedge \ldots \wedge \mathcal{S}_a[\![P_n]\!] \wedge \mathcal{S}_d[\![P]\!] \Longrightarrow \mathcal{S}_a[\![P]\!]$$

3. specifica *dettagliata* $\mathcal{S}_e[\![P]\!]$;

3a. verifica specifica dettagliata: $\mathcal{S}_e[\![P]\!] \Longrightarrow \mathcal{S}_a[\![P]\!]$



C-Tilco: Comunicazione basso livello (2)

Caratteristiche comunicazione:

- ritardo d costante maggiore o uguale a zero;
- il canale non crea messaggi:

$$p_{in}.\text{receive}(m) \wedge p_{out}\text{->}d\text{>-}p_{in} \longrightarrow p_{out}.\text{send}(m)\,@\,[-d,-d]$$

- canale senza perdita:

$$p_{out}.\text{send}(m) \wedge p_{out}\text{->}d\text{>-}p_{in} \longrightarrow p_{in}.\text{receive}(m)\,@\,[d,d]$$

- invio e ricezione di un messaggio alla volta:

$$p_{out}.\text{send}(m_1) \wedge p_{out}.\text{send}(m_2) \longrightarrow m_1 = m_2$$
$$p_{in}.\text{receive}(m_1) \wedge p_{in}.\text{receive}(m_2) \longrightarrow m_1 = m_2$$

C-Tilco: Comunicazione basso livello (1)

Si occupa dell'invio e della ricezione di informazioni e messaggi di controllo tramite le porte di comunicazione.

- p_{out} porta di uscita:
  - p_{out}.send(msg) predicato che indica l'invio del messaggio msg tramite la porta p_{out}
  - p_{out}.receiveAck predicato che indica la ricezione di un Ack tramite la porta p_{out}
- p_{in} porta di ingresso:
  - p_{in}.receive(msg) predicato che indica la ricezione del messaggio msg tramite la porta p_{in}
  - p_{in}.sendAck predicato che indica l'invio di un Ack tramite la porta p_{in}
- collegamento: $p_{out}\text{->}d\text{>-}p_{in}$ predicato che indica il collegamento della porta di uscita con la porta di ingresso con ritardo d.

C-Tilco: Comunicazione Sincrona (2)

Regole *Emissione*:

- *Send rule*

$$p_o \,!!\, m\,[W_s]\,;;\,P_s \quad \longrightarrow \quad p_o.\text{send}\, m \wedge \text{until}_0\,(p_o.receiveAck \wedge P_s)(\neg p_o.\text{receiveAck} \wedge W_s)) \wedge (p_o.\text{receiveAck} \vee (\neg p_o.\text{receiveAck} \wedge \text{until}\,(p_o.\text{receiveAck}\,)(\neg p_o.\text{receiveAck} \wedge \neg p_o!!)))$$

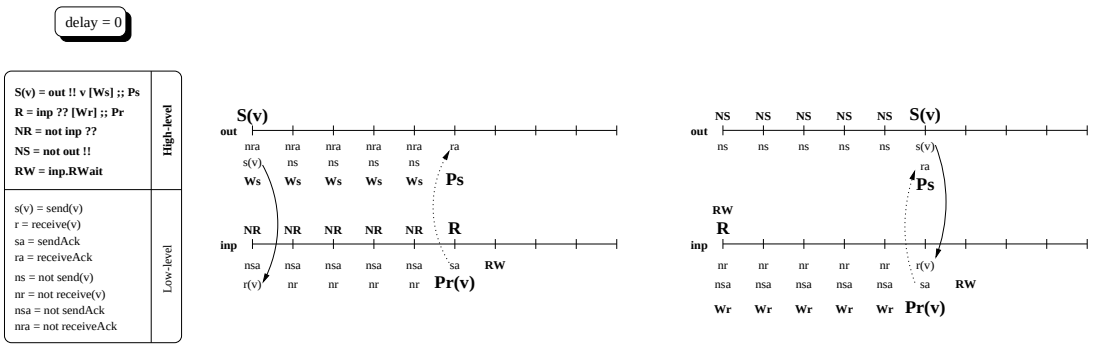
- *noSend rule*

$$\neg p_o\,!! \longrightarrow \forall m.\, \neg p_o.\text{send}\, m$$

C-Tilco: Comunicazione Sincrona (1)

Operatori:

- Send: $p_{out}\,!!\,m\,[W_s]\,;;\,P_s$
- Receive: $p_{in}\,??\,[W_r]\,;;\,P_r$
- noSend: $\neg p_{out}\,!!$
- noReceive: $\neg p_{in}\,??$



C-Tilco: Teoremi Comunicazione Sincrona (1)

Receive-until0D

$$\frac{\text{now}\,(p_i\,??\,[W_r]\,;;\,P_r)\;t}{\text{now}\,(\exists v.\;\text{until}_0\;P_r(v)\;W_r)\;t}$$

Receive-until0'D

$$\frac{\text{now}\,(p_i\,??\,[W_r]\,;;\,P_r)\;t \quad \text{now}\,(\exists k.\;p_i.\text{receive}\,(k)?[0,+\infty))\;t}{\text{now}\,(\exists v.\;\text{until}'_0\;P_r(v)\;W_r)\;t}$$

Send-until0D

$$\frac{\text{now}\,(p_o\;!!\;v\;[W_s]\;;;\;P_s)\;t}{\text{now}\,(\text{until}_0\;P_s\;W_s)\;t}$$

Send-until0'D

$$\frac{\text{now}\,(p_o\;!!\;v\;[W_s]\;;;\;P_s)\;t \quad \dfrac{\text{now}\,(p_o.\text{send}\,(v))\;t}{\text{now}\,(p_o.\text{receiveAck}\,?[0,+\infty))\;t}}{\text{now}\,(\text{until}'_0\;P_s\;W_s)\;t}$$

C-Tilco: Comunicazione Sincrona (3)

Regole *Ricezione*:

- *RValue def* $\quad p_i.\text{RValue}\,v \equiv \text{since}'\,(p_i.\text{receive}\,v \wedge \neg p_i.\text{sendAck}\,)(\neg p_i.\text{sendAck}\,)$
- *RWait def* $\quad p_i.\text{RWait} \equiv \neg\exists m.\;p_i.\text{RValue}\,m$
- *Receive-RValue rule*

$$(p_i\,??\,[W_r]\,;;\,P_r) \wedge p_i.\text{RValue}\,m \longrightarrow p_i.\text{sendAck} \wedge P_r(m)$$

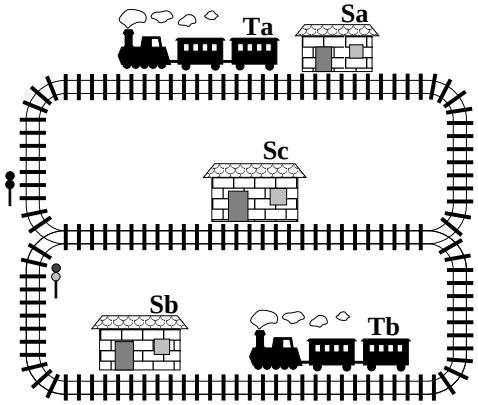
- *Receive-RWait rule*

$$\begin{aligned}&(p_i\,??\,[W_r]\,;;\,P_r) \wedge p_i.\text{RWait} \longrightarrow\\ &\quad \text{until}_0\;(\exists m.\;p_i.\text{receive}\,m \wedge p_i.\text{sendAck} \wedge P_r(m))(\neg(\exists m.\;p_i.\text{receive}\,m) \wedge W_r)) \wedge\\ &\quad ((\exists m.\;p_i.\text{receive}\,m) \vee (\neg(\exists m.\;p_i.\text{receive}\,m) \wedge \neg p_i.\text{sendAck} \wedge\\ &\qquad \text{until}\,(\exists m.\;p_i.\text{receive}\,m)(\neg(\exists m.\;p_i.\text{receive}\,m) \wedge \neg p_i??)))\end{aligned}$$

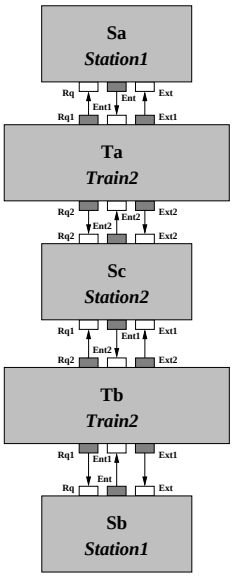
- *noReceive rule*

$$\neg p_i\,?? \longrightarrow \neg p_i.\text{sendAck}$$

C-Tilco: Esempio (1)



Decomposizione sistema



C-Tilco: Teoremi Comunicazione Sincrona (2)

RWait-Receive-then-SendE

$$\frac{\text{now } p_i.\text{RWait } t \qquad \begin{array}{l} p_o \xrightarrow{0} p_i \\ \text{now } (p_i \,??\, [W_r] \,;;\, P_r)\ t \\ \text{now } (p_o \,!!\, v\ [W_s] \,;;\, P_s)\ (s+t) \\ \text{now } (\neg p_o !! @ [0,s))\ t \end{array} \qquad \frac{\begin{array}{l} \text{now } P_r(v)\ (s+t) \\ \text{now } P_s\ (s+t) \\ \text{now } (W_r @ [0,s))\ t \\ \text{now } p_i.\text{RWait}\ (s+t+1) \end{array}}{R} \quad 0 \leq s}{R}$$

RWait-Send-then-ReceiveE

$$\frac{\text{now } p_i.\text{RWait } t \qquad \begin{array}{l} p_o \xrightarrow{0} p_i \\ \text{now } (p_o \,!!\, v\ [W_s] \,;;\, P_s)\ t \\ \text{now } (p_i \,??\, [W_r] \,;;\, P_r)\ (s+t) \\ \text{now } (\neg p_o ?? @ [0,s))\ t \end{array} \qquad \frac{\begin{array}{l} \text{now } P_r(v)\ (s+t) \\ \text{now } P_s\ (s+t) \\ \text{now } (W_s @ [0,s))\ t \\ \text{now } p_i.\text{RWait}\ (s+t+1) \end{array}}{R} \quad 0 \leq s}{R}$$

RWait-noSend-noReceiveE

$$\frac{\text{now } p_i.\text{RWait } t \qquad \begin{array}{l} p_o \xrightarrow{0} p_i \\ \text{now } (\neg p_o !! @ [0,s))\ t \\ \text{now } (\neg p_o ?? @ [0,s))\ t \end{array} \qquad \frac{\text{now } p_i.\text{RWait}\ (s+t+1)}{R} \quad 0 \leq s}{R}$$

C-Tilco: Esempio (3)

Specifica *Station2*

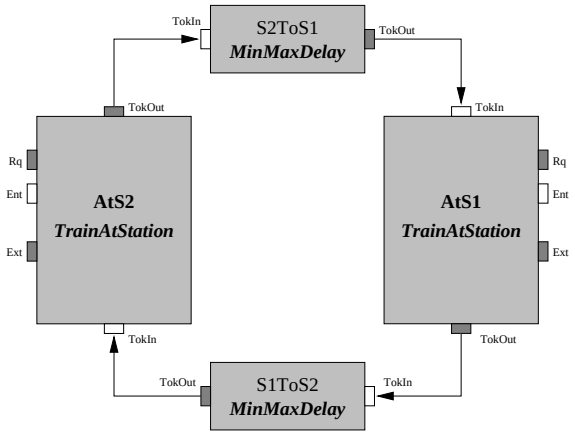
- `Rq1`/`Rq2` porte di ingresso per la ricezione della richiesta di accesso da parte del primo/secondo treno;
- `Ent1`/`Ent2` porte di uscita su cui inviare la conferma di accesso alla stazione per il primo/secondo treno;
- `Ext1`/`Ext2` porte di ingresso per la ricezione della notifica di uscita dalla stazione del primo/secondo treno;
- `hasTrain1`/`hasTrain2` predicati che indicano se il primo/secondo treno è nella stazione;

In ogni istante nella stazione non ci possono essere entrambi i treni:

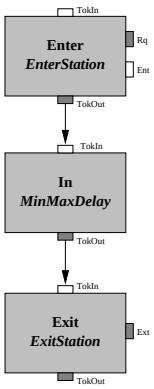
$$(\neg(\texttt{hasTrain1} \wedge \texttt{hasTrain2}))@(-\infty, +\infty)$$

C-Tilco: Esempio (2)

Decomposizione *Train2*



Decomposizione *TrainAtStation*



C-Tilco: Esempio (5)

Specifica dettagliata *Station2*

$$\texttt{free} \Longleftrightarrow \neg\texttt{hasTrain1} \wedge \neg\texttt{hasTrain2}$$

$$\texttt{process\_start} \Longrightarrow \texttt{waitRq1} \wedge \texttt{waitRq2} \wedge \texttt{free} \,@\, (-\infty, 0]$$

$$\texttt{waitRq1} \Longrightarrow \texttt{Rq1} \,??\, [\neg\texttt{req1} \wedge \neg\texttt{hasTrain1}] \,;;\, (\lambda\, m.\, \texttt{req1} \wedge \neg\texttt{hasTrain1})$$
$$\texttt{waitRq2} \Longrightarrow \texttt{Rq2} \,??\, [\neg\texttt{req2} \wedge \neg\texttt{hasTrain2}] \,;;\, (\lambda\, m.\, \texttt{req2} \wedge \neg\texttt{hasTrain2})$$

$$\texttt{free} \wedge \neg\texttt{req1} \wedge \neg\texttt{req2} \;\twoheadrightarrow\; \texttt{free}$$
$$\texttt{free} \wedge \texttt{req1} \wedge \neg\texttt{req2} \;\twoheadrightarrow\; \texttt{sendEnt1}$$
$$\texttt{free} \wedge \texttt{req2} \wedge \neg\texttt{req1} \;\twoheadrightarrow\; \texttt{sendEnt2}$$

$$\texttt{free} \wedge \texttt{req1} \wedge \texttt{req2} \twoheadrightarrow \texttt{sendEnt1} \wedge \texttt{req2} \wedge \neg\texttt{hasTrain2}$$

$$\neg\texttt{free} \wedge \texttt{req1} \;\twoheadrightarrow\; \texttt{req1} \wedge \neg\texttt{hasTrain1}$$
$$\neg\texttt{free} \wedge \texttt{req2} \;\twoheadrightarrow\; \texttt{req2} \wedge \neg\texttt{hasTrain2}$$

$$\texttt{sendEnt1} \;\Rightarrow\; \texttt{Ent1} \,!!\, m\, [\texttt{hasTrain1} \wedge \neg\texttt{req1}] \,;;\, \texttt{waitExt1}$$
$$\texttt{sendEnt2} \;\Rightarrow\; \texttt{Ent2} \,!!\, m\, [\texttt{hasTrain2} \wedge \neg\texttt{req2}] \,;;\, \texttt{waitExt2}$$

$$\texttt{waitExt1} \;\Rightarrow\; \texttt{Ext1} \,??\, [\texttt{hasTrain1} \wedge \neg\texttt{req1}] \,;;\, (\lambda\, v.\, \neg\texttt{hasTrain1} \wedge \neg\texttt{req1} \wedge \texttt{waitRq1} \,@\, [1, 1])$$
$$\texttt{waitExt2} \;\Rightarrow\; \texttt{Ext2} \,??\, [\texttt{hasTrain2} \wedge \neg\texttt{req2}] \,;;\, (\lambda\, v.\, \neg\texttt{hasTrain2} \wedge \neg\texttt{req2} \wedge \texttt{waitRq2} \,@\, [1, 1])$$

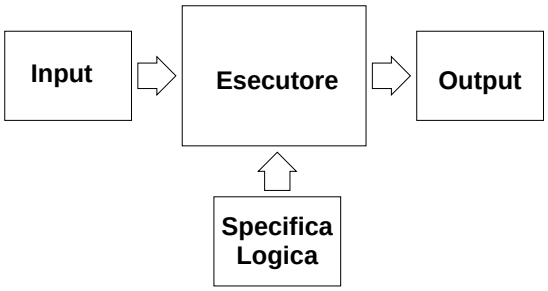
C-Tilco: Esempio (4)

Specifica *Station2*

Predicati ausiliari per la specifica dettagliata:

- `free` predicato che indica se la stazione è libera;
- `waitRq1`/`waitRq2` predicati che indicano di iniziare ad attendere la richiesta da parte del primo/secondo treno;
- `req1`/`req2` predicati che indicano la presenza di una richiesta da parte del primo treno, questi predicati rimangono veri finchè il treno non ha accesso alla stazione;
- `sendEnt1`/`sendEnt2` predicati che indicano di inviare al primo/secondo treno la conferma di accesso alla stazione e quindi di attendere l'arrivo della notifica di uscita dalla stazione;
- `waitExt1`/`waitExt2` predicati che indicano di iniziare ad attendere l'arrivo della notifica di uscita dalla stazione del primo/secondo treno;

Esecuzione di specifiche logiche



Logica BTL

Operatori: $A \wedge B, A \vee B, \neg A, \Delta^k A, \bowtie A$

$\Delta^k \equiv$ ritardo di k istanti

$\bowtie \equiv$ è sempre vero che

TILCO-X

Estensione di TILCO

- Intervalli Dinamici

$$\text{Start} \rightarrow \neg\text{End} \,@\, [0, +30) \wedge \text{End} \,?\, [+30, +\text{Start})$$

$$\text{until}\, A\ B \equiv B \,@\, (0, +A)$$

$$\text{since}\, A\ B \equiv B \,@\, (-A, 0)$$

- Bounded Happen

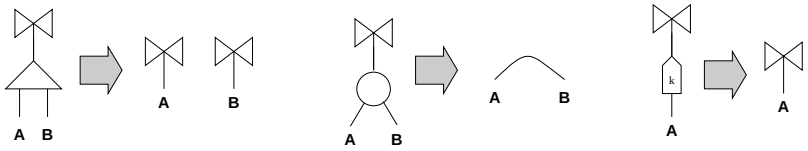
$A \,?_3^5\, [0, 100)$ A vero da 3 a 5 volte in [0,100)

$A \,?^5\, [0, +B)$ A vero al massimo 5 volte in [0,+B)

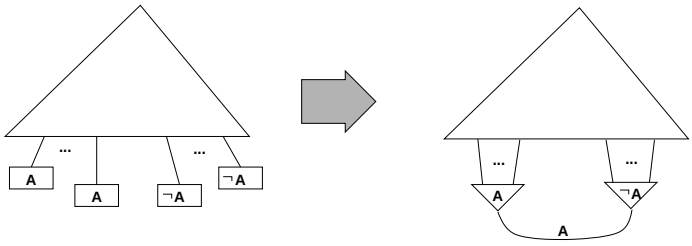
- Nuovo sistema deduttivo.

BTL → TIN

1. espansione operatori non elementari $(A \rightarrow B \equiv \neg A \vee B,\ A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A))$
2. $\neg$ portato al livello più basso;
3. costruzione albero sintattico;
4. semplificazione

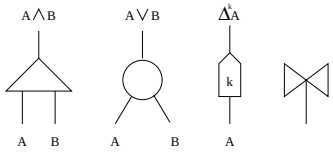


5. chiusura segnali

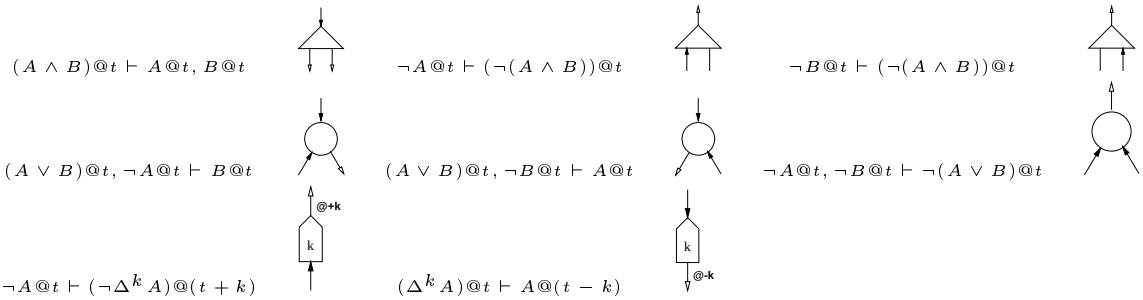


Reti di Inferenza Temporale (TIN)

Elementi di base:



Regole di inferenza:



Esempio

Espressione BTL:

$$\bowtie (up \leftrightarrow (\Delta \neg l \wedge l))$$

Rete TIN: