

Industrial Standards, Computer Algebra, and Formal Verification

Dominik Dietrich Lutz Schröder **Ewaryst Schulz**

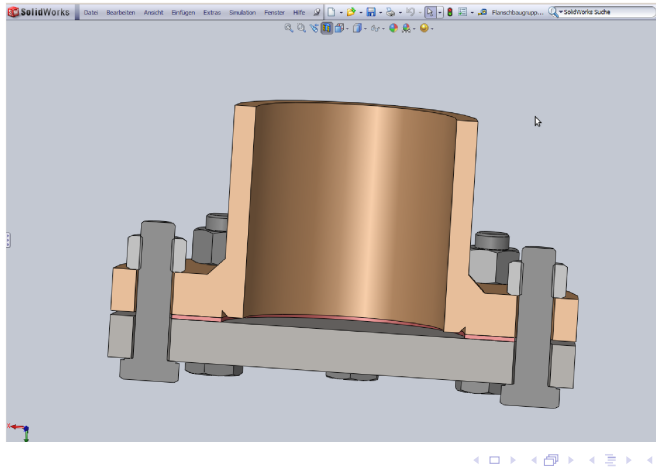
DFKI Bremen, Germany
ewaryst.schulz@dfki.de

20th International Workshop on Algebraic Development Techniques
Schloss Etelsen, Germany
4th July 2010

The Flange



A CAD design of a **flange-bolt-gasket system**.



The Industrial Standard EN 1591



A standard for **gasketed circular flange connections**

The standard consists of

► **Applicability** and basic assumptions

► **Nomenclature**

► **Calculation method**

The calculation method **assures** the **impermeability** and **mechanical strength** of the flange-bolt-gasket system.

DEUTSCHE NORM		Oktober 2001
Flansche und ihre Verbindungen Regeln für die Auslegung von Flanschverbindungen mit runden Flanschen und Dichtung Teil 1: Berechnungsmethode Deutsche Fassung EN 1591-1:2001		DIN EN 1591-1
ICS 23.040.60		Mit DIN V EN 1591-2:2001-10 Ersatz für DIN V 2505:1986-01
Flanges and their joints — Design rules for gasketed circular flange connections — Part 1: Calculation method; German version EN 1591-1:2001		
Brides et leurs assemblages — Règles de calcul des assemblages à brides circulaires avec joint — Partie 1: Méthode de calcul; Version allemande EN 1591-1:2001		
Die Europäische Norm EN 1591-1:2001 hat den Status einer Deutschen Norm.		
Nationales Vorwort		
Diese Europäische Norm ist vom Technischen Komitee CEN/TC 74 "Flansche und ihre Verbindungen" (Sekretariat: DIN) unter deutscher Mitwirkung ausgearbeitet worden.		
Für die deutsche Mitarbeit ist der Arbeitsausschuss NARD-74 "Flansche und ihre Verbindungen" im Normenausschuss Rohrleitungen und Dampfkesselanlagen (NARD) verantwortlich.		
Anhang A bis Anhang G und Anhang ZA sind informativ.		
Hintergrundinformationen zu DIN EN 1591-1 sind im Beiblatt 1 zu DIN EN 1591-1 enthalten.		
Änderungen		
Gegenüber DIN V 2505:1986-01 wurden folgende Änderungen vorgenommen:		
– Die Europäische Norm EN 1591-1 basiert auf der Berechnungsmethode nach TGL 32 903/13:1983.		
– Vormomcharakter aufgehoben		
Frühere Ausgaben		
DIN 2506:1927-01, 1964-10 DIN V 2505: 1986-01 DIN 2506: 1927-01 DIN 2507: 1927-07		
Normenausschuss Rohrleitungen und Dampfkesselanlagen (NARD) im DIN Deutschen Institut für Normung e. V.		

© DIN Deutsches Institut für Normung e. V. Jede Art der Vervielfältigung, auch auszugsweise,
ist mit Genehmigung des DIN Deutsches Institut für Normung e. V., Berlin, gestattet.
Alleinvertauf der Normen durch Beuth Verlag GmbH, 10772 Berlin.

PM: Nr. DIN EN 1591-1:2001-10
Page: 17 von 26 2017

The Industrial Standard EN 1591



A standard for **gasketed circular flange connections**

The standard consists of

- ▶ **Applicability** and basic assumptions
- ▶ **Nomenclature**
- ▶ **Calculation method**

The calculation method **assures** the **impermeability** and **mechanical strength** of the flange-bolt-gasket system.

DEUTSCHE NORM		Oktober 2001
Flansche und ihre Verbindungen Regeln für die Auslegung von Flanschverbindungen mit runden Flanschen und Dichtung Teil 1: Berechnungsmethode Deutsche Fassung EN 1591-1:2001		DIN EN 1591-1
ICS 23.040.60		Mit DIN V EN 1591-2:2001-10 Ersatz für DIN V 2505:1986-01
Flanges and their joints — Design rules for gasketed circular flange connections — Part 1: Calculation method; German version EN 1591-1:2001		
Brides et leurs assemblages — Règles de calcul des assemblages à brides circulaires avec joint — Partie 1: Méthode de calcul; Version allemande EN 1591-1:2001		
Die Europäische Norm EN 1591-1:2001 hat den Status einer Deutschen Norm.		
Nationales Vorwort		
Diese Europäische Norm ist vom Technischen Komitee CEN/TC 74 "Flansche und ihre Verbindungen" (Sekretariat: DIN) unter deutscher Mitwirkung ausgearbeitet worden.		
Für die deutsche Mitarbeit ist der Arbeitsausschuss NARD-74 "Flansche und ihre Verbindungen" im Normenausschuss Rohrleitungen und Dampfkesselanlagen (NARD) verantwortlich.		
Anhang A bis Anhang G und Anhang ZA sind informativ.		
Hintergrundinformationen zu DIN EN 1591-1 sind im Beiblatt 1 zu DIN EN 1591-1 enthalten.		
Änderungen		
Gegenüber DIN V 2505:1986-01 wurden folgende Änderungen vorgenommen:		
– Die Europäische Norm EN 1591-1 basiert auf der Berechnungsmethode nach TGL 32 903/13:1983.		
– Vormomcharakter aufgehoben		
Frühere Ausgaben		
DIN 2506:1927-01, 1964-10 DIN V 2505: 1986-01 DIN 2506: 1927-01 DIN 2507: 1927-07		
Normenausschuss Rohrleitungen und Dampfkesselanlagen (NARD) im DIN Deutschen Institut für Normung e. V.		

© DIN Deutsches Institut für Normung e. V. Jede Art der Vervielfältigung, auch auszugsweise,
nur mit Genehmigung des DIN Deutsches Institut für Normung e. V., Berlin, gestattet.
Alleinvertauf der Normen durch Beuth Verlag GmbH, 10772 Berlin.

PDF Nr.: DIN EN 1591-1:2001-10
Preis: 17,- Euro, ab 2017



The Industrial Standard EN 1591



A standard for gasketed circular flange connections

The standard consists of

- ▶ **Applicability** and basic assumptions
- ▶ **Nomenclature**
- ▶ **Calculation method**

The calculation method assures the **impermeability** and **mechanical strength** of the flange-bolt-gasket system.

EN 1591-1:2001

6.4 Integrierter Flansch, Bund oder Bördel

Auslastungsgrad für Flansch, Bund oder Bördel (für Bund oder Bördel gilt $\Phi_{\text{inn}} = 1,0$):

$$\Phi_0 = F_Z \times b_0 + F_L \times (b_0 - b_L) + F_M \times b_M / W_Z \leq \Phi_{\text{adm}} \quad (75)$$

$$W_Z = (\pi/4) \times [f_b \times 2 \times b_0 \times e_0^2 \times (1 + 2 \times \Psi_{\text{ort}} \times \Psi_Z^2) + f_L \times d_L \times e_0^2 \times c_M \times J_M \times k_M] \quad (76)$$

$$e_0 = e_1 \times \left[1 + \frac{(\beta - 1) \times l_M}{\sqrt{(8/3)^4 \times (d_1 \times e_1)^2 + l_M^2}} \right] \quad (75)$$

$$f_b = \min(f_c, f_d) \quad (76)$$

$$\delta_0 = P \times d_0 / (f_b \times 2 \times e_0 \times \cos\Phi_0); \quad \delta_M = F_M / (f_L \times \pi \times d_L \times e_0 \times \cos\Phi_0) \quad (77)$$

$$e_0 = \begin{cases} \sqrt{\frac{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,75 \times \delta_0 + 1 + \delta_0^2]}{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,25 \times \delta_0 + 3 + \delta_0^2]}} & \text{für Kegel- und Zylinderschalen} \\ \sqrt{\frac{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,75 \times \delta_0 + 1 + \delta_0^2]}{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,25 \times \delta_0 + 3 + \delta_0^2]}} & \text{für Kegel- und Zylinderschalen} \end{cases} \quad (78)$$

$$e_1 = \begin{cases} \sqrt{\frac{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,75 \times \delta_0 + 1 + \delta_0^2]}{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,25 \times \delta_0 + 3 + \delta_0^2]}} & \text{für Kegel- und Zylinderschalen} \\ \sqrt{\frac{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,75 \times \delta_0 + 1 + \delta_0^2]}{1,33 \times [1 - 0,75 \times (0,5 \times \delta_0 + \delta_M)] \times [1 - 0,25 \times \delta_0 + 3 + \delta_0^2]}} & \text{für Kegel- und Zylinderschalen} \end{cases} \quad (79)$$

$$J_M = \pi b_M^3 (F_Z \times b_0 + F_L \times (b_0 - b_L) + F_M \times b_M); \quad j_M = 1 \quad (80)$$

$$-1 \leq k_M \leq +1; \quad 0 \leq \delta_0 \leq 1 \quad (81)$$

$$\Psi_{\text{ort}} = \begin{cases} \frac{f_c \times d_c \times e_0 \times \cos\Phi_0}{f_b \times 2 \times b_0 \times e_0} \\ \times \left[0,5 \times \delta_0 + \delta_M \times \cos\Phi_0 - \delta_0 \times 2 \times e_0 \delta_c \times J_M \times j_M \times \sqrt{\frac{e_0 \times \cos\Phi_0}{d_c \times \cos\Phi_0}} \right] \quad (82) \end{cases}$$

$$W_Z = (\pi/4) \times [f_b \times 2 \times b_0 \times e_0^2 \times (1 + 2 \times \Psi_{\text{ort}} \times \Psi_Z^2) + f_L \times d_L \times e_0^2 \times c_M \times J_M \times k_M]$$

$$e_0 = e_1 \times \left[1 + \frac{(\beta - 1) \times l_M}{\sqrt{(8/3)^4 \times (d_1 \times e_1)^2 + l_M^2}} \right]$$

$$f_b = \min(f_c, f_d)$$

$$\delta_0 = P \times d_0 / (f_b \times 2 \times e_0 \times \cos\Phi_0); \quad \delta_M = F_M / (f_L \times \pi \times d_L \times e_0 \times \cos\Phi_0)$$

The Industrial Standard EN 1591



A standard for **gasketed circular flange connections**

The standard consists of

- ▶ **Applicability** and basic assumptions
- ▶ **Nomenclature**
- ▶ **Calculation method**

The calculation method **assures** the **impermeability** and **mechanical strength** of the flange-bolt-gasket system.

EN 1591-1:2001

6.4 Integrierter Flansch, Bund oder Bördel

Auslastungsgrad für Flansch, Bund oder Bördel (für Bund oder Bördel gilt $\Phi_{\text{min}} = 1,0$):

$$\Phi_0 = F_D \times b_D + F_L \times (b_D - b_L) + F_K \times b_M / W_D \leq \Phi_{\text{max}} \quad (75)$$

$$W_D = (\pi/4) \times [f_D \times 2 \times b_D \times e_D^2 \times (1 + 2 \times \Psi_{\text{opt}} \times \Psi_Z^2) + F_L \times d_L \times e_D^2 \times c_M \times J_M \times b_M] \quad (76)$$

$$e_D = e_1 \times \left[1 + \frac{(\beta - 1) \times l_M}{\sqrt{(\beta/3)^4 \times (d_L \times e_1)^2 + l_M^2}} \right] \quad (75)$$

$$f_D = \min(f_D, f_D) \quad (76)$$

$$\delta_D = P \times d_D / (f_D \times 2 \times e_D \times \cos\Phi_0); \quad \delta_K = F_M / (f_K \times \pi \times d_K \times e_D \times \cos\Phi_0) \quad (77)$$

$$e_D = \begin{cases} \sqrt{\frac{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,75 \times \delta_D + 1 + \delta_D]}{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,25 \times \delta_D + 0,5 \times \delta_D]}} & \text{für Kegel- und Zylinderschalen} \\ \sqrt{\frac{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,75 \times \delta_D + 1 + \delta_D]}{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,25 \times \delta_D + 0,5 \times \delta_D]}} & \text{für Kegel- und Zylinderschalen} \end{cases} \quad (78)$$

$$e_D = \begin{cases} \sqrt{\frac{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,75 \times \delta_D + 1 + \delta_D]}{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,25 \times \delta_D + 0,5 \times \delta_D]}} & \text{für Kegel- und Zylinderschalen} \\ \sqrt{\frac{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,75 \times \delta_D + 1 + \delta_D]}{0,33 \times [1 - 0,75 \times (0,5 \times \delta_D + \delta_D)] \times [1 - 0,25 \times \delta_D + 0,5 \times \delta_D]}} & \text{für Kegel- und Zylinderschalen} \end{cases} \quad (79)$$

$$J_M = \delta_M \times [F_D \times b_D + F_L \times (b_D - b_L) + F_K \times b_M]; \quad j_M = 1 \quad (80)$$

$$-1 \leq j_M \leq +1; \quad 0 \leq \delta_D \leq 1 \quad (81)$$

$$\Psi_{\text{opt}} = \frac{f_D \times e_D \times c_M \times \cos\Phi_0}{f_D \times 2 \times b_D \times e_D} \times \left[\frac{c_M \times e_D \times \tau_{0,2} \times \tau_{0,1} \times [1 + J_M \times b_M]}{d_L \times \cos\Phi_0} \right] \quad (82)$$

$$W_D = (\pi/4) \times [f_D \times 2 \times b_D \times e_D^2 \times (1 + 2 \times \Psi_{\text{opt}} \times \Psi_Z^2) + F_L \times d_L \times e_D^2 \times c_M \times J_M \times b_M]$$

$$e_D = e_1 \times \left[1 + \frac{(\beta - 1) \times l_M}{\sqrt{(\beta/3)^4 \times (d_L \times e_1)^2 + l_M^2}} \right]$$

$$f_D = \min(f_D, f_D)$$

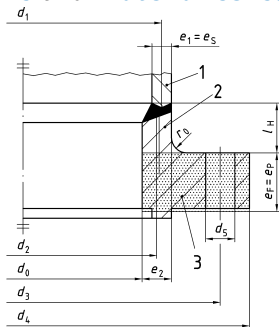
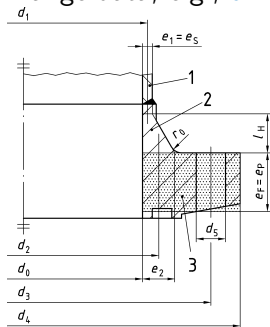
$$\delta_D = P \times d_D / (f_D \times 2 \times e_D \times \cos\Phi_0); \quad \delta_K = F_M / (f_K \times \pi \times d_K \times e_D \times \cos\Phi_0)$$

The Industrial Standard EN 1591 cont.



The **input parameters** to the calculation method

- ▶ Flange data, e.g., **dimensions** and **material constants**



- ▶ **Mounting data** such as screw tightening method
- ▶ Data for **operating states** such as pressure and temperature



$$b_{G0} = \left\{ \begin{array}{ll} (d_{3c} - d_{Ge})/2 & \text{für Integralfansch und Blindfansch} \\ (d_{70} - d_{Ge})/2 & \text{für Losfansch und Bund oder Bördel} \end{array} \right\} \quad (40)$$

$$d_{70} = \min \left\{ \max (d_{7min}; (d_{Ge} + \mathbf{x} \times d_{3c})/(1 + \mathbf{x}); d_{7max}) \right\} \quad (41)$$

$$\mathbf{x} = (Z_1 \times E_{F0})/(Z_F \times E_{1,0})$$

ANMERKUNG 3 Gleichung (41) gilt nur für lose Flansche mit Bördel oder Bund.

Gleichungen (38) bis (41) werden so lange iterativ ausgewertet, bis der Wert b_{Ge} innerhalb der erforderlichen Genauigkeit konstant bleibt.

ANMERKUNG 4 Eine Genauigkeit von 5 % ist ausreichend. Um ein Ergebnis zu erreichen, das vom Anwender unabhängig ist, wird eine Genauigkeit von 0,1 % empfohlen.

Tabelle 1 — Effektive Dichtungsgeometrie

Typ	Dichtungsform	Formeln
1	Flachdichtungen, Weichstoff oder zusammengesetzte Werkstoffe oder reines Metall, siehe Bild 3 a	<p>Erste Näherung:</p> $b_{Gi} = b_{Gi}$ <p>Genauer:</p> $b_{Gi} = \sqrt{\frac{e_d/(\pi \times d_{Ge} \times E_{Gm})}{h_{G0} \times Z_F/E_{F0} + \bar{h}_{G0} \times \bar{Z}_F/\bar{E}_{F0}} + \left[\frac{F_{G0}}{\pi \times d_{Ge} \times Q_{max,y}} \right]^2}$ $E_{Gm} = E_0 + 0,5 K_1 \times F_{G0}/A_{Ge}$ $Z_F, \bar{Z}_F \text{ nach Gleichung (27) oder (31)}$ <p>Immer: $d_{Ge} = d_{G2} - b_{Ge}$</p>



$$h_{G0} = \left\{ \begin{array}{ll} (d_{3e} - d_{Ge})/2 & \text{für Integralfansch und Blindfansch} \\ (d_{70} - d_{Ge})/2 & \text{für Losfansch und Bund oder Bördel} \end{array} \right\} \quad (40)$$

$$\left. \begin{array}{l} d_{70} = \min \{ \max (d_{7min}; (d_{Ge} + \mathbf{x} \times d_{3e})/(1 + \mathbf{x}); d_{7max}) \\ \mathbf{x} = (Z_1 \times E_{F0})/(Z_F \times E_{1.0}) \end{array} \right\} \quad (41)$$

ANMERKUNG 3 Gleichung (41) gilt nur für lose Flansche mit Bördel oder Bund.

Gleichungen (38) bis (41) werden so lange iterativ ausgewertet, bis der Wert b_{Gi} innerhalb der erforderlichen Genauigkeit konstant bleibt.

ANMERKUNG 4 Eine Genauigkeit von 5 % ist ausreichend. Um ein Ergebnis zu erreichen, das vom Anwender unabhängig ist, wird eine Genauigkeit von 0,1 % empfohlen.

Tabelle 1 — Effektive Dichtungsgeometrie

Typ	Dichtungsform	Formeln
1	Flachdichtungen, Weichstoff oder zusammengesetzte Werkstoffe oder reines Metall, siehe Bild 3 a	<p>Erste Näherung:</p> $b_{Gi} = b_{Gi}$ <p>Genauer:</p> $b_{Gi} = \sqrt{\frac{e_d/(\pi \times d_{Ge} \times E_{Gm})}{h_{G0} \times Z_F/E_{F0} + \bar{h}_{G0} \times \bar{Z}_F/\bar{E}_{F0}} + \left[\frac{F_{G0}}{\pi \times d_{Ge} \times Q_{max,y}} \right]^2}$ <p>$E_{Gm} = E_0 + 0,5 K_1 \times F_{G0}/A_{Ge}$</p> <p>$Z_F, \bar{Z}_F$ nach Gleichung (27) oder (31)</p> <p>Immer: $d_{Ge} = d_{G2} - b_{Ge}$</p>





$$b_{G0} = \left\{ \begin{array}{ll} (d_{3c} - d_{Ge})/2 & \text{für Integralfansch und Blindfansch} \\ (d_{70} - d_{Ge})/2 & \text{für Losfansch und Bund oder Bördel} \end{array} \right\} \quad (40)$$

$$d_{70} = \min \left\{ \max (d_{7min}; (d_{Ge} + \mathbf{x} \times d_{3c})/(1 + \mathbf{x}); d_{7max}) \right\} \quad (41)$$

$$\mathbf{x} = (Z_1 \times E_{F0})/(Z_F \times E_{10})$$

ANMERKUNG 3 Gleichung (41) gilt nur für lose Flansche mit Bördel oder Bund.

Gleichungen (38) bis (41) werden so lange iterativ ausgewertet, bis der Wert b_{Ge} innerhalb der erforderlichen Genauigkeit konstant bleibt.

ANMERKUNG 4 Eine Genauigkeit von 5 % ist ausreichend. Um ein Ergebnis zu erreichen, das vom Anwender unabhängig ist, wird eine Genauigkeit von 0,1 % empfohlen.

Tabelle 1 — Effektive Dichtungsgeometrie

Typ	Dichtungsform	Formeln
1	Flachdichtungen, Weichstoff oder zusammengesetzte Werkstoffe oder reines Metall, siehe Bild 3 a	<p>Erste Näherung:</p> $b_{Gi} = b_{Gi}$ <p>Genauer:</p> $b_{Gi} = \sqrt{\frac{e_d/(\pi \times d_{Ge} \times E_{Gm})}{h_{G0} \times Z_F/E_{F0} + \bar{h}_{G0} \times \bar{Z}_F/\bar{E}_{F0}} + \left[\frac{F_{G0}}{\pi \times d_{Ge} \times Q_{max,y}} \right]^2}$ $E_{Gm} = E_0 + 0,5 K_1 \times F_{G0}/A_{Ge}$ $Z_F, \bar{Z}_F \text{ nach Gleichung (27) oder (31)}$ <p>Immer: $d_{Ge} = d_{G2} - b_{Ge}$</p>



$$b_{G0} = \left\{ \begin{array}{ll} (d_{3c} - d_{Ge})/2 & \text{für Integralfansch und Blindfansch} \\ (d_{70} - d_{Ge})/2 & \text{für Losfansch und Bund oder Bördel} \end{array} \right\} \quad (40)$$

$$d_{70} = \min \left\{ \max (d_{7min}; (d_{Ge} + \mathbf{x} \times d_{3c})/(1 + \mathbf{x}); d_{7max}) \right\} \quad (41)$$

$$\mathbf{x} = (Z_1 \times E_{F0})/(Z_F \times E_{10})$$

ANMERKUNG 3 Gleichung (41) gilt nur für lose Flansche mit Bördel oder Bund.

Gleichungen (38) bis (41) werden so lange iterativ ausgewertet, bis der Wert b_{Gi} innerhalb der erforderlichen Genauigkeit konstant bleibt.

ANMERKUNG 4 Eine Genauigkeit von 5 % ist ausreichend. Um ein Ergebnis zu erreichen, das vom Anwender unabhängig ist, wird eine Genauigkeit von 0,1 % empfohlen.

Tabelle 1 — Effektive Dichtungsgeometrie

Typ	Dichtungsform	Formeln
1	Flachdichtungen, Weichstoff oder zusammengesetzte Werkstoffe oder reines Metall, siehe Bild 3 a	<p>Erste Näherung:</p> $b_{Gi} = b_{G0}$ <p>Genauer:</p> $b_{Gi} = \sqrt{\frac{e_d/(\pi \times d_{Ge} \times E_{Gm})}{h_{G0} \times Z_F/E_{F0} + \bar{h}_{G0} \times \bar{Z}_F/\bar{E}_{F0}} + \left[\frac{F_{G0}}{\pi \times d_{Ge} \times Q_{max,y}} \right]^2}$ $E_{Gm} = E_0 + 0,5 K_1 \times F_{G0}/A_{Ge}$ $Z_F, \bar{Z}_F \text{ nach Gleichung (27) oder (31)}$ <p>Immer: $d_{Ge} = d_{G2} - b_{Ge}$</p>



$$W_F = (\pi/4) \times \{f_F \times 2 \times b_F \times e_F^2 \times (1 + 2 \times \Psi_{opt} \times \Psi_Z - \Psi_Z^2) + f_E \times d_E \times e_D^2 \times c_M \times j_M \times k_M\} \quad (74)$$

$j_M = +1$	$\Psi_{max} \leq \Psi_{opt}$	$k_M = +1$	$\Psi_Z = \Psi_{max}$
	$\Psi_0 \leq \Psi_{opt} < \Psi_{max}$	$k_M = +1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_{opt} < \Psi_0$	$k_M < +1$	$\Psi_Z = \Psi_{(-1, k_M, +1)}$
$j_M = -1$	$\Psi_{opt} \leq \Psi_{min}$	$k_M = -1$	$\Psi_Z = \Psi_{min}$
	$\Psi_{min} < \Psi_{opt} \leq \Psi_0$	$k_M = -1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_0 < \Psi_{opt}$	$k_M > -1$	$\Psi_Z = \Psi_{(+1, k_M, +1)}$

Der Rechnungsgang ist wie folgt durchzuführen:

- e_D wird nach Gleichung (75) nach Bestimmung von β durch Auswerten von Gleichung (9) errechnet.
- $f_E, \delta_Q, \delta_R, c_M$ werden nach den Gleichungen (76) bis (78) berechnet.
(Wird der Wert unter der Wurzel von c_M negativ, ist der Hals überlastet).
- Es werden $c_{S(j_S = +1)}, c_{S(j_S = -1)}, j_M, \Psi_{opt}, \Psi_0, \Psi_{max}, \Psi_{min}$ nach den Gleichungen (79) bis (84) errechnet.
(Wenn $\Psi_{max} < -1$ oder $\Psi_{min} > +1$, ist der Ring überlastet).
- k_M und Ψ_Z sind nach Tabelle 2 auszuwählen. Wenn die Tabelle $k_M < +1$ oder $k_M > -1$ ohne weitere Präzisierung festlegt, muss der Wert von k_M derart abgestimmt werden, dass W_F in Gleichung (74) maximal wird nach Berechnung im folgenden Schritt e). Der Wert von Ψ_Z zugeordnet zu k_M ist durch Gleichung (82) gegeben.



$$W_F = (\pi/4) \times \{f_F \times 2 \times b_F \times e_F^2 \times (1 + 2 \times \Psi_{opt} \times \Psi_Z - \Psi_Z^2) + f_E \times d_E \times e_D^2 \times c_M \times j_M \times k_M\} \quad (74)$$

$j_M = +1$	$\Psi_{max} \leq \Psi_{opt}$	$k_M = +1$	$\Psi_Z = \Psi_{max}$
	$\Psi_0 \leq \Psi_{opt} < \Psi_{max}$	$k_M = +1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_{opt} < \Psi_0$	$k_M < +1$	$\Psi_Z = \Psi_{(-1, k_M, +1)}$
$j_M = -1$	$\Psi_{opt} \leq \Psi_{min}$	$k_M = -1$	$\Psi_Z = \Psi_{min}$
	$\Psi_{min} < \Psi_{opt} \leq \Psi_0$	$k_M = -1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_0 < \Psi_{opt}$	$k_M > -1$	$\Psi_Z = \Psi_{(+1, k_M, +1)}$

Der Rechnungsgang ist wie folgt durchzuführen:

- e_D wird nach Gleichung (75) nach Bestimmung von β durch Auswerten von Gleichung (9) errechnet.
- f_E , δ_Q , δ_R , c_M werden nach den Gleichungen (76) bis (78) berechnet.
(Wird der Wert unter der Wurzel von c_M negativ, ist der Hals überlastet).
- Es werden $c_{S(j_S = +1)}$, $c_{S(j_S = -1)}$, j_M , Ψ_{opt} , Ψ_0 , Ψ_{max} , Ψ_{min} nach den Gleichungen (79) bis (84) errechnet.
(Wenn $\Psi_{max} < -1$ oder $\Psi_{min} > +1$, ist der Ring überlastet).
- k_M und Ψ_Z sind nach Tabelle 2 auszuwählen. Wenn die Tabelle $k_M < +1$ oder $k_M > -1$ ohne weitere Präzisierung festlegt, muss der Wert von k_M derart abgestimmt werden, dass W_F in Gleichung (74) maximal wird nach Berechnung im folgenden Schritt e). Der Wert von Ψ_Z zugeordnet zu k_M ist durch Gleichung (82) gegeben.



$$W_F = (\pi/4) \times \{f_F \times 2 \times b_F \times e_F^2 \times (1 + 2 \times \Psi_{opt} \times \Psi_Z - \Psi_Z^2) + f_E \times d_E \times e_D^2 \times c_M \times j_M \times k_M\} \quad (74)$$

$j_M = +1$	$\Psi_{max} \leq \Psi_{opt}$	$k_M = +1$	$\Psi_Z = \Psi_{max}$
	$\Psi_0 \leq \Psi_{opt} < \Psi_{max}$	$k_M = +1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_{opt} < \Psi_0$	$k_M < +1$	$\Psi_Z = \Psi_{(-1, k_M, +1)}$
$j_M = -1$	$\Psi_{opt} \leq \Psi_{min}$	$k_M = -1$	$\Psi_Z = \Psi_{min}$
	$\Psi_{min} < \Psi_{opt} \leq \Psi_0$	$k_M = -1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_0 < \Psi_{opt}$	$k_M > -1$	$\Psi_Z = \Psi_{(+1, k_M, +1)}$

Der Rechnungsgang ist wie folgt durchzuführen:

- e_D wird nach Gleichung (75) nach Bestimmung von β durch Auswerten von Gleichung (9) errechnet.
- $f_E, \delta_Q, \delta_R, c_M$ werden nach den Gleichungen (76) bis (78) berechnet.
(Wird der Wert unter der Wurzel von c_M negativ, ist der Hals überlastet).
- Es werden $c_{S(j_S = +1)}, c_{S(j_S = -1)}, j_M, \Psi_{opt}, \Psi_0, \Psi_{max}, \Psi_{min}$ nach den Gleichungen (79) bis (84) errechnet.
(Wenn $\Psi_{max} < -1$ oder $\Psi_{min} > +1$, ist der Ring überlastet).
- k_M und Ψ_Z sind nach Tabelle 2 auszuwählen. Wenn die Tabelle $k_M < +1$ oder $k_M > -1$ ohne weitere Präzisierung festlegt, muss der Wert von k_M derart abgestimmt werden, dass W_F in Gleichung (74) maximal wird nach Berechnung im folgenden Schritt e). Der Wert von Ψ_Z zugeordnet zu k_M ist durch Gleichung (82) gegeben.



$$W_F = (\pi/4) \times \{f_F \times 2 \times b_F \times e_F^2 \times (1 + 2 \times \Psi_{opt} \times \Psi_Z - \Psi_Z^2) + f_E \times d_E \times e_D^2 \times c_M \times j_M \times k_M\} \quad (74)$$

$j_M = +1$	$\Psi_{max} \leq \Psi_{opt}$	$k_M = +1$	$\Psi_Z = \Psi_{max}$
	$\Psi_0 \leq \Psi_{opt} < \Psi_{max}$	$k_M = +1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_{opt} < \Psi_0$	$k_M < +1$	$\Psi_Z = \Psi_{(-1, k_M, +1)}$
$j_M = -1$	$\Psi_{opt} \leq \Psi_{min}$	$k_M = -1$	$\Psi_Z = \Psi_{min}$
	$\Psi_{min} < \Psi_{opt} \leq \Psi_0$	$k_M = -1$	$\Psi_Z = \Psi_{opt}$
	$\Psi_0 < \Psi_{opt}$	$k_M > -1$	$\Psi_Z = \Psi_{(+1, k_M, +1)}$

Der Rechnungsgang ist wie folgt durchzuführen:

- e_D wird nach Gleichung (75) nach Bestimmung von β durch Auswerten von Gleichung (9) errechnet.
- f_E , δ_Q , δ_R , c_M werden nach den Gleichungen (76) bis (78) berechnet.
(Wird der Wert unter der Wurzel von c_M negativ, ist der Hals überlastet).
- Es werden $c_{S(j_S = +1)}$, $c_{S(j_S = -1)}$, j_M , Ψ_{opt} , Ψ_0 , Ψ_{max} , Ψ_{min} nach den Gleichungen (79) bis (84) errechnet.
(Wenn $\Psi_{max} < -1$ oder $\Psi_{min} > +1$, ist der Ring überlastet).
- k_M und Ψ_Z sind nach Tabelle 2 auszuwählen. Wenn die Tabelle $k_M < +1$ oder $k_M > -1$ ohne weitere Präzisierung festlegt, muss der Wert von k_M derart abgestimmt werden, dass W_F in Gleichung (74) maximal wird nach Berechnung im folgenden Schritt e). Der Wert von Ψ_Z zugeordnet zu k_M ist durch Gleichung (82) gegeben.

Calculation Method and Computer Algebra



The **formulas** occurring in the standard can be **calculated** using

- ▶ Standard **real arithmetic**
- ▶ **Real functions** such as \cos , $\sqrt[n]{}$, etc.
- ▶ **Special functions** such as *maximize*
- ▶ **Control structures** such as conditional statements and iteration

Use a **computer algebra system** for the calculations.

Calculation Method and Computer Algebra



The **formulas** occurring in the standard can be **calculated** using

- ▶ Standard **real arithmetic**
- ▶ **Real functions** such as \cos , $\sqrt[n]{}$, etc.
- ▶ **Special functions** such as *maximize*
- ▶ **Control structures** such as conditional statements and iteration

Use a **computer algebra system** for the calculations.

Calculation Method and Computer Algebra



The **formulas** occurring in the standard can be **calculated** using

- ▶ Standard **real arithmetic**
- ▶ **Real functions** such as \cos , $\sqrt[n]{}$, etc.
- ▶ **Special functions** such as *maximize*
- ▶ **Control structures** such as conditional statements and iteration

Use a **computer algebra system** for the calculations.

Calculation Method and Computer Algebra



The **formulas** occurring in the standard can be **calculated** using

- ▶ Standard **real arithmetic**
- ▶ **Real functions** such as \cos , $\sqrt[n]{}$, etc.
- ▶ **Special functions** such as *maximize*
- ▶ **Control structures** such as conditional statements and iteration

Use a **computer algebra system** for the calculations.

Calculation Method and Computer Algebra



The **formulas** occurring in the standard can be **calculated** using

- ▶ Standard **real arithmetic**
- ▶ **Real functions** such as \cos , $\sqrt[n]{}$, etc.
- ▶ **Special functions** such as *maximize*
- ▶ **Control structures** such as conditional statements and iteration

Use a **computer algebra system** for the calculations.



Correctness of calculations crucial for application to safety critical environments

- ▶ CASs do **not** provide **justifications** of calculations
- ▶ $\frac{x}{x}$ simplifies to 1 in the Reduce CAS

Results of the CAS can be formally **verified**

- ▶ One can **generate lemmas** from CAS result to be proved
- ▶ **Checking** is **easier** than finding



Correctness of calculations crucial for application to safety critical environments

- ▶ CASs do **not** provide **justifications** of calculations
- ▶ $\frac{x}{x}$ simplifies to 1 in the Reduce CAS

Results of the CAS can be formally **verified**

- ▶ One can **generate lemmas** from CAS result to be proved
- ▶ **Checking** is **easier** than finding



Correctness of calculations crucial for application to safety critical environments

- ▶ CASs do **not** provide **justifications** of calculations
- ▶ $\frac{x}{x}$ simplifies to 1 in the Reduce CAS

Results of the CAS can be formally **verified**

- ▶ One can **generate lemmas** from CAS result to be proved
- ▶ **Checking** is **easier** than finding



Correctness of calculations crucial for application to safety critical environments

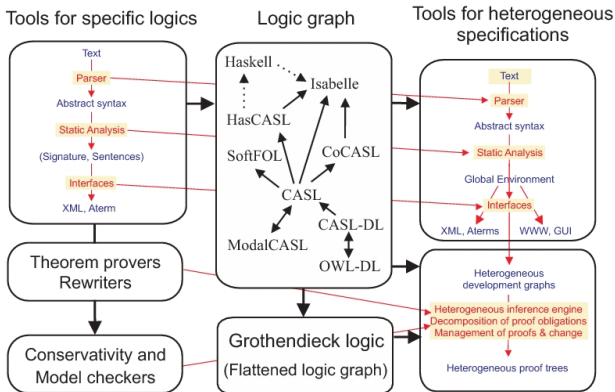
- ▶ CASs do **not** provide **justifications** of calculations
- ▶ $\frac{x}{x}$ simplifies to 1 in the Reduce CAS

Results of the CAS can be formally **verified**

- ▶ One can **generate lemmas** from CAS result to be proved
- ▶ **Checking** is **easier** than finding



Architecture of the heterogeneous tool set Hets





Design goals of CSL

- ▶ Formal **specification** of the **calculation method**
- ▶ Specification of assignments in an **arbitrary order**, but:
We require assignments to be **unique and sortable** w.r.t. the dependency order
- ▶ Generic interface to CAS

Translation to CAS

- ▶ Suitably ordered **assignments** together with control structures form an **imperative program**
- ▶ Constants depending on constants which were **modified** are **recomputed**
- ▶ Executing the program using CAS yields a **symbolic valuation**





Design goals of CSL

- ▶ Formal **specification** of the **calculation method**
- ▶ Specification of assignments in an **arbitrary order**, but:
We require assignments to be **unique and sortable** w.r.t. the dependency order
- ▶ Generic interface to CAS

Translation to CAS

- ▶ Suitably ordered **assignments** together with control structures form an **imperative program**
- ▶ Constants depending on constants which were **modified** are **recomputed**
- ▶ Executing the program using CAS yields a **symbolic valuation**





Design goals of CSL

- ▶ Formal **specification** of the **calculation method**
- ▶ Specification of assignments in an **arbitrary order**, but:
We require assignments to be **unique and sortable** w.r.t. the dependency order
- ▶ Generic interface to CAS

Translation to CAS

- ▶ Suitably ordered **assignments** together with control structures form an **imperative program**
- ▶ Constants depending on constants which were **modified** are **recomputed**
- ▶ Executing the program using CAS yields a **symbolic valuation**





Design goals of CSL

- ▶ Formal **specification** of the **calculation method**
- ▶ Specification of assignments in an **arbitrary order**, but:
We require assignments to be **unique and sortable** w.r.t. the dependency order
- ▶ Generic interface to CAS

Translation to CAS

- ▶ Suitably ordered **assignments** together with control structures form an **imperative program**
- ▶ Constants depending on constants which were **modified** are **recomputed**
- ▶ Executing the program using CAS yields a **symbolic valuation**





Design goals of CSL

- ▶ Formal **specification** of the **calculation method**
- ▶ Specification of assignments in an **arbitrary order**, but:
We require assignments to be **unique and sortable** w.r.t. the dependency order
- ▶ Generic interface to CAS

Translation to CAS

- ▶ Suitably ordered **assignments** together with control structures form an **imperative program**
- ▶ Constants depending on constants which were **modified** are **recomputed**
- ▶ Executing the program using CAS yields a **symbolic valuation**



Design goals of CSL

- ▶ Formal **specification** of the **calculation method**
- ▶ Specification of assignments in an **arbitrary order**, but:
We require assignments to be **unique and sortable** w.r.t. the dependency order
- ▶ Generic interface to CAS

Translation to CAS

- ▶ Suitably ordered **assignments** together with control structures form an **imperative program**
- ▶ Constants depending on constants which were **modified** are **recomputed**
- ▶ Executing the program using CAS yields a **symbolic valuation**





Calculating a root of \cos using **Newton's Method**

The CSL specification

```
y := cos(x) %(A)%  
z := sin(x) %(B)%  
x := 10 %(C)%  
repeat  
  x := x + y/z %(D)%  
until abs(y) < 0.001
```

The **translation** yields this **program**:

```
C;A;B;repeat D;A;B; until abs(y) < 0.001
```

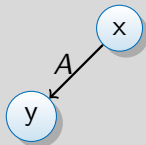


Calculating a root of \cos using **Newton's Method**

The CSL specification

```
y := cos(x) %(A)%  
z := sin(x) %(B)%  
x := 10 %(C)%  
repeat  
  x := x + y/z %(D)%  
until abs(y) < 0.001
```

Building the Dependency Graph



The **translation** yields this **program**:

```
C;A;B;repeat D;A;B; until abs(y) < 0.001
```

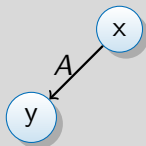


Calculating a root of \cos using **Newton's Method**

The CSL specification

```
y := cos(x) %(A)%  
z := sin(x) %(B)%  
x := 10 %(C)%  
repeat  
  x := x + y/z %(D)%  
until abs(y) < 0.001
```

Building the Dependency Graph



The **translation** yields this **program**:

```
C;A;B;repeat D;A;B; until abs(y) < 0.001
```



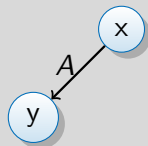


Calculating a root of \cos using **Newton's Method**

The CSL specification

```
y := cos(x) %(A)%  
z := sin(x) %(B)%  
x := 10 %(C)%  
repeat  
  x := x + y/z %(D)%  
until abs(y) < 0.001
```

Building the Dependency Graph



The **translation** yields this **program**:

```
C;A;B;repeat D;A;B; until abs(y) < 0.001
```

$\downarrow C$

$\swarrow B$



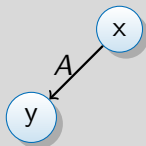


Calculating a root of \cos using **Newton's Method**

The CSL specification

```
y := cos(x) %(A)%  
z := sin(x) %(B)%  
x := 10 %(C)%  
repeat  
  x := x + y/z %(D)%  
until abs(y) < 0.001
```

Building the Dependency Graph



The **translation** yields this **program**:

```
C;A;B;repeat D;A;B; until abs(y) < 0.001
```

C



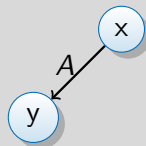


Calculating a root of \cos using **Newton's Method**

The CSL specification

```
y := cos(x) %(A)%  
z := sin(x) %(B)%  
x := 10 %(C)%  
repeat  
  x := x + y/z %(D)%  
until abs(y) < 0.001
```

Building the Dependency Graph



The **translation** yields this **program**:

```
C;A;B;repeat D;A;B; until abs(y) < 0.001
```





Verification Points in CSL

- ▶ are **positions of subterms** of CSL statements
- ▶ Evaluating a such marked term produces a **verification condition**
- ▶ The **CAS result** is **extended** by a list of verification conditions
- ▶ Use Hets to **prove verification conditions**

Specifying CAS program semantics in HasCASL

- ▶ Standard interpretation of programs as **state transformers**
- ▶ Properties of **algorithms** specified in CSL can be **verified**



Verification Points in CSL

- ▶ are **positions of subterms** of CSL statements
- ▶ Evaluating a such marked term produces a **verification condition**
- ▶ The **CAS result** is **extended** by a list of verification conditions
- ▶ Use Hets to **prove verification conditions**

Specifying CAS program semantics in HasCASL

- ▶ Standard interpretation of programs as **state transformers**
- ▶ Properties of **algorithms** specified in CSL can be **verified**



Verification Points in CSL

- ▶ are **positions of subterms** of CSL statements
- ▶ Evaluating a such marked term produces a **verification condition**
- ▶ The **CAS result** is **extended** by a list of verification conditions
- ▶ Use Hets to **prove verification conditions**

Specifying CAS program semantics in HasCASL

- ▶ Standard interpretation of programs as **state transformers**
- ▶ Properties of **algorithms** specified in CSL can be **verified**



Verification Points in CSL

- ▶ are **positions of subterms** of CSL statements
- ▶ Evaluating a such marked term produces a **verification condition**
- ▶ The **CAS result** is **extended** by a list of verification conditions
- ▶ Use Hets to **prove verification conditions**

Specifying CAS program semantics in HasCASL

- ▶ Standard interpretation of programs as **state transformers**
- ▶ Properties of **algorithms** specified in CSL can be **verified**



Verification Points in CSL

- ▶ are **positions of subterms** of CSL statements
- ▶ Evaluating a such marked term produces a **verification condition**
- ▶ The **CAS result** is **extended** by a list of verification conditions
- ▶ Use Hets to **prove verification conditions**

Specifying CAS program semantics in HasCASL

- ▶ Standard interpretation of programs as **state transformers**
- ▶ Properties of **algorithms** specified in CSL can be **verified**



Verification Points in CSL

- ▶ are **positions of subterms** of CSL statements
- ▶ Evaluating a such marked term produces a **verification condition**
- ▶ The **CAS result** is **extended** by a list of verification conditions
- ▶ Use Hets to **prove verification conditions**

Specifying CAS program semantics in HasCASL

- ▶ Standard interpretation of programs as **state transformers**
- ▶ Properties of **algorithms** specified in CSL can be **verified**



Verifying a result from the CAS

A CAS program

```
⋮ Environment =  $\sigma$   
y := maximize(t, x)  
⋮
```

- ▶ We set verification point at maximize position \rightarrow maximize(t, x) is marked
- ▶ CAS computes this expression in context σ and returns result r
- ▶ Apply substitution σ to t and obtain t'
- ▶ We produce the verification condition $\text{maximize}(t', x) = r$
- ▶ Translate this equality to HasCASL for proving



Verifying a result from the CAS

A CAS program

```
⋮ Environment =  $\sigma$   
y := maximize(t, x)  
⋮
```

- ▶ We set verification point at maximize position \rightarrow maximize(t, x) is marked
- ▶ CAS computes this expression in context σ and returns result r
- ▶ Apply substitution σ to t and obtain t'
- ▶ We produce the verification condition $maximize(t', x) = r$
- ▶ Translate this equality to HasCASL for proving



Verifying a result from the CAS

A CAS program

```
⋮ Environment = σ  
y := maximize(t, x)  
⋮
```

- ▶ We set verification point at maximize position → `maximize(t, x)` is marked
- ▶ CAS computes this expression in context σ and returns result r
- ▶ Apply substitution σ to t and obtain t'
- ▶ We produce the verification condition $\text{maximize}(t', x) = r$
- ▶ Translate this equality to HasCASL for proving



Verifying a result from the CAS

A CAS program

```
⋮ Environment =  $\sigma$   
y := maximize(t, x)  
⋮
```

- ▶ We set verification point at maximize position \rightarrow maximize(t, x) is marked
- ▶ CAS computes this expression in context σ and returns result r
- ▶ Apply substitution σ to t and obtain t'
- ▶ We produce the verification condition $maximize(t', x) = r$
- ▶ Translate this equality to HasCASL for proving



Verifying a result from the CAS

A CAS program

```
⋮ Environment = σ  
y := maximize(t, x)  
⋮
```

- ▶ We set verification point at maximize position → `maximize(t, x)` is marked
- ▶ CAS computes this expression in context σ and returns result r
- ▶ Apply substitution σ to t and obtain t'
- ▶ We produce the verification condition $\text{maximize}(t', x) = r$
- ▶ Translate this equality to HasCASL for proving



Verifying a result from the CAS

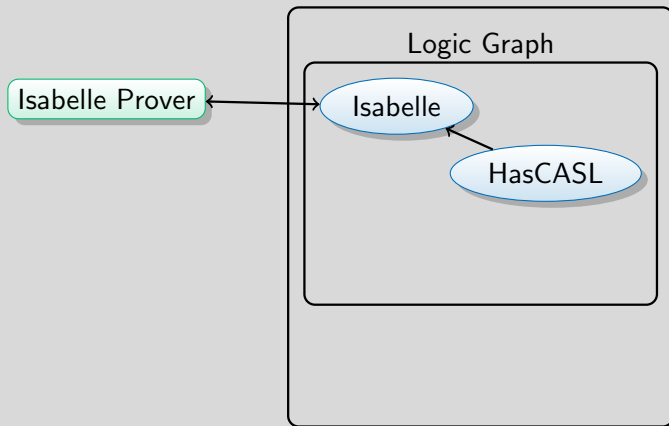
A CAS program

```
⋮ Environment = σ  
y := maximize(t, x)  
⋮
```

- ▶ We set verification point at maximize position \rightarrow maximize(t , x) is marked
- ▶ CAS computes this expression in context σ and returns result r
- ▶ Apply substitution σ to t and obtain t'
- ▶ We produce the verification condition $maximize(t', x) = r$
- ▶ Translate this equality to HasCASL for proving

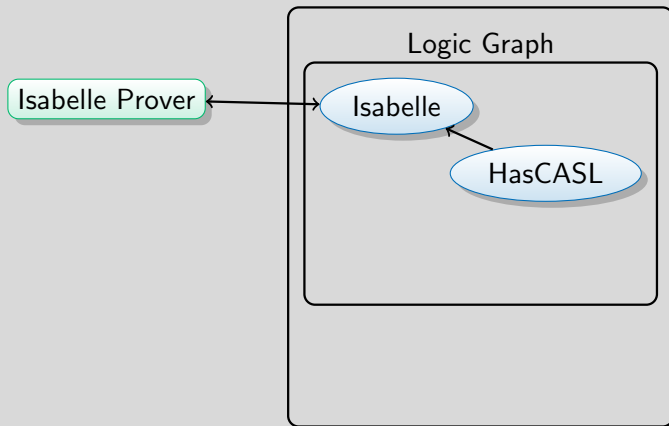


CSL and the Hets Logic Graph



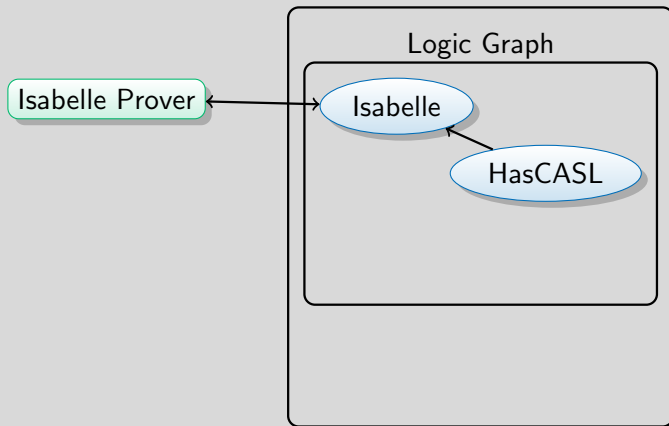


CSL and the Hets Logic Graph



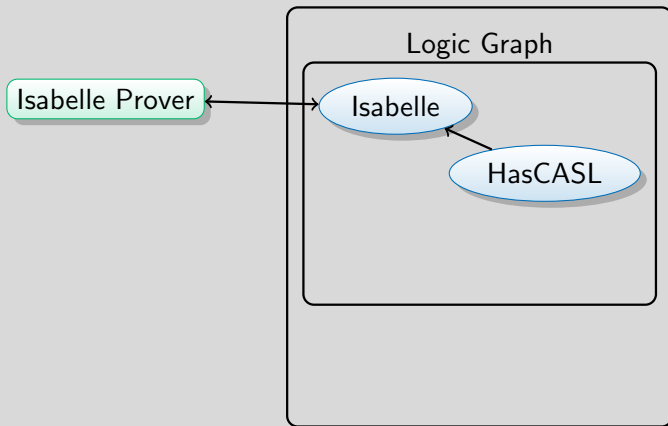


CSL and the Hets Logic Graph





CSL and the Hets Logic Graph





The CSL institution

- ▶ **Signatures** are collections of real constants and functions over the reals
- ▶ **Sentences** are program statements or first order formulas in an extended theory of the reals augmented by the signature
- ▶ **Models** are program states, i.e., symbolic valuations
- ▶ A state **satisfies a program** if it terminates successfully
- ▶ A state **satisfies a formula** ϕ if ϕ holds under this valuation



The CSL institution

- ▶ **Signatures** are collections of real constants and functions over the reals
- ▶ **Sentences** are program statements or first order formulas in an extended theory of the reals augmented by the signature
- ▶ **Models** are program states, i.e., symbolic valuations
- ▶ A state **satisfies a program** if it terminates successfully
- ▶ A state **satisfies a formula** ϕ if ϕ holds under this valuation



The CSL institution

- ▶ **Signatures** are collections of real constants and functions over the reals
- ▶ **Sentences** are program statements or first order formulas in an extended theory of the reals augmented by the signature
- ▶ **Models** are program states, i.e., symbolic valuations
 - ▶ A state **satisfies a program** if it terminates successfully
 - ▶ A state **satisfies a formula** ϕ if ϕ holds under this valuation



The CSL institution

- ▶ **Signatures** are collections of real constants and functions over the reals
- ▶ **Sentences** are program statements or first order formulas in an extended theory of the reals augmented by the signature
- ▶ **Models** are program states, i.e., symbolic valuations
- ▶ A state **satisfies a program** if it terminates successfully
- ▶ A state **satisfies a formula** ϕ if ϕ holds under this valuation



The CSL institution

- ▶ **Signatures** are collections of real constants and functions over the reals
- ▶ **Sentences** are program statements or first order formulas in an extended theory of the reals augmented by the signature
- ▶ **Models** are program states, i.e., symbolic valuations
- ▶ A state **satisfies a program** if it terminates successfully
- ▶ A state **satisfies a formula** ϕ if ϕ holds under this valuation



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard



- ▶ **Specification language** CSL for industrial standards
- ▶ **Synthesis of programs** for generic CAS interface
- ▶ **Verification Points** for local verification of CAS result
- ▶ **Integration** of CSL and CAS interface in Hets
- ▶ Specification of **CSL semantics** in HasCASL
- ▶ **Relating CSL to HasCASL** by theoroidal comorphism

Benefit from symbolic character of CAS computations

- ▶ Using CAS to **simplify** CSL specifications for partial instantiations or given set of additional assumptions
- ▶ Replace special functions by **closed solutions** found by the CAS
- ▶ **Finding instantiations** for underspecified specifications, e.g., number of bolts needed for flange to satisfy standard